Model Driven Security
Foundations, Tools, and Practice

David Basin, Manuel Clavel, Marina Egea
ETH Zurich and IMDEA Software

Module Objectives

Present a methodology and tools for automatically building secure, complex, distributed applications.

Formal: Has a well defined mathematical semantics.

General: Ideas may be specialized in many ways.

Usable: Based on familiar concepts and notation.

Wide spectrum: Integrates security into overall design process.

Tool supported: Compatible too with UML-based design tools.

Scales: Initial experience (academic and industry) positive.

Overall Structure

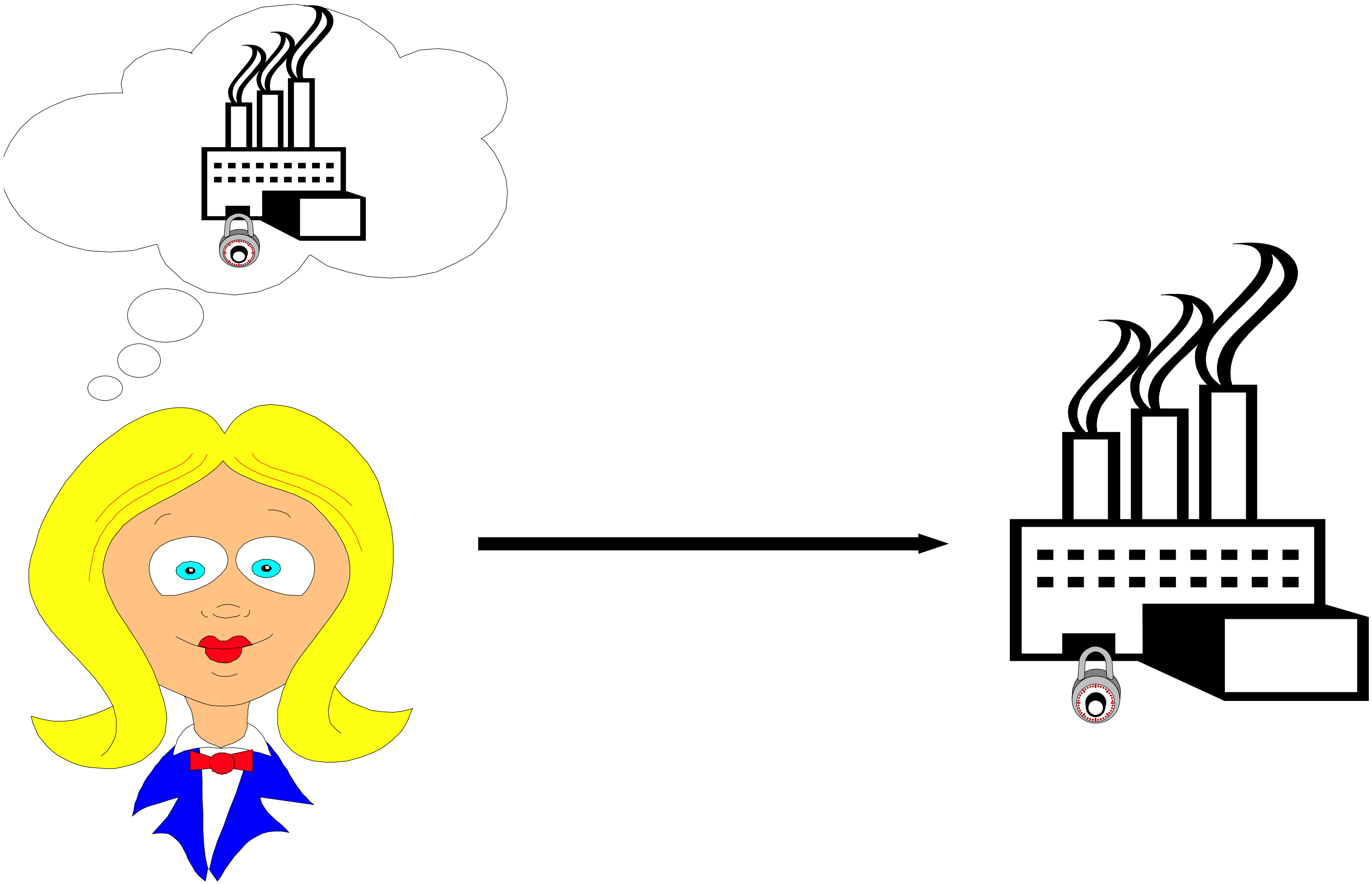
1. Basic Ideas
   - Component and process models
   - Security models
   - Combination
   - Generation
2. Extensions
   - GUI models
   - Database integration
   - Model analysis
   - Model transformation: security at different levels
3. Tool Support
4. Case Studies

Key: DB: 1, ME: 1–3, MC: 3–4

Road Map (this talk)

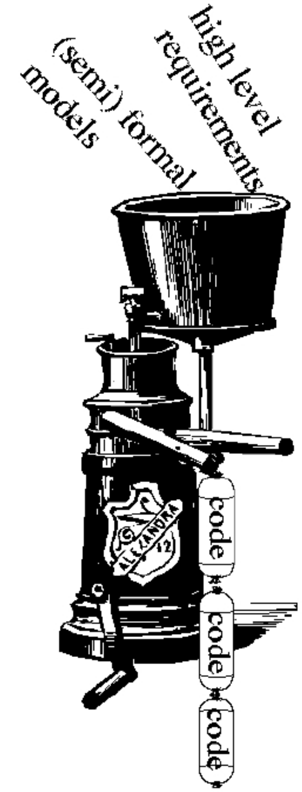
- Motivation and objectives
- Background
- Secure components
- Semantics
- Generating security infrastructures
- Secure controllers
- Experience and conclusions

Motivation



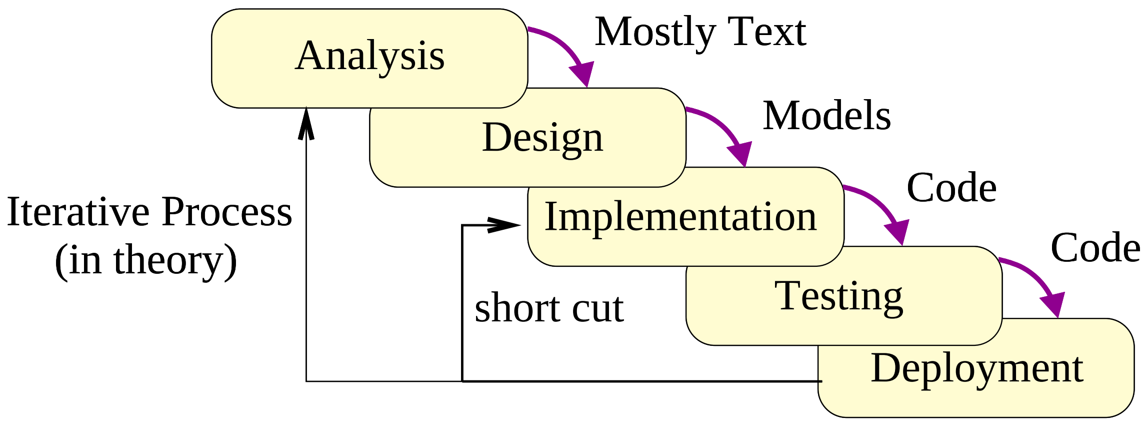
How do we go from requirements to secure systems?

From Requirements to Systems



- Ideally: Automated synthesis from specifications.
  - The Holy Grail of Software Engineering!
  - But problem is not recursively solvable.

- As described by process models.

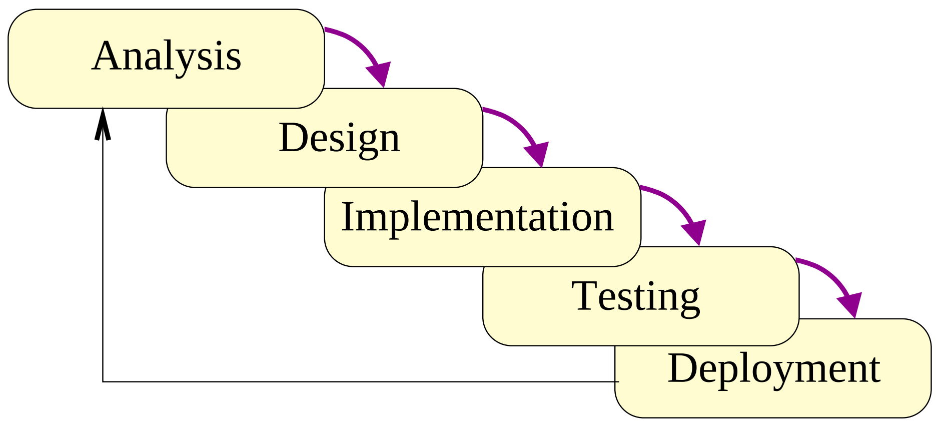


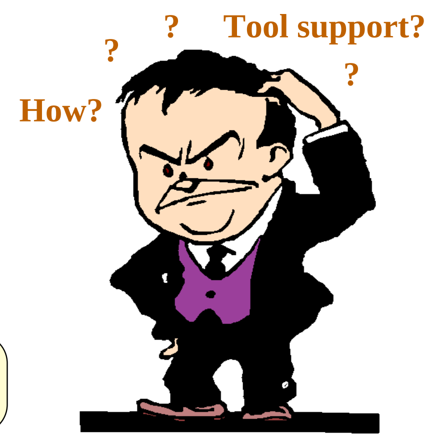
- In practice: code-and-fix.

  Adequate in-the-small. But poor quality control and scalability.

From Requirements to Systems: Security

- Engineering security into system design is usually neglected.





- Ad hoc integration has a negative impact on security.

- Two gaps to bridge:

Requirements Analysis
Security Policies

Implementation
Design Models

Running Example: A Meeting Scheduler

Functional requirements:

System should maintain a list of users and records of meetings. A meeting has an owner, a list of participants, a time, and a place. Users may carry out operations on meetings such as creating, reading, editing, and deleting them. A user may also cancel a meeting, which deletes the meeting and notifies all participants by email ...

Security requirements:

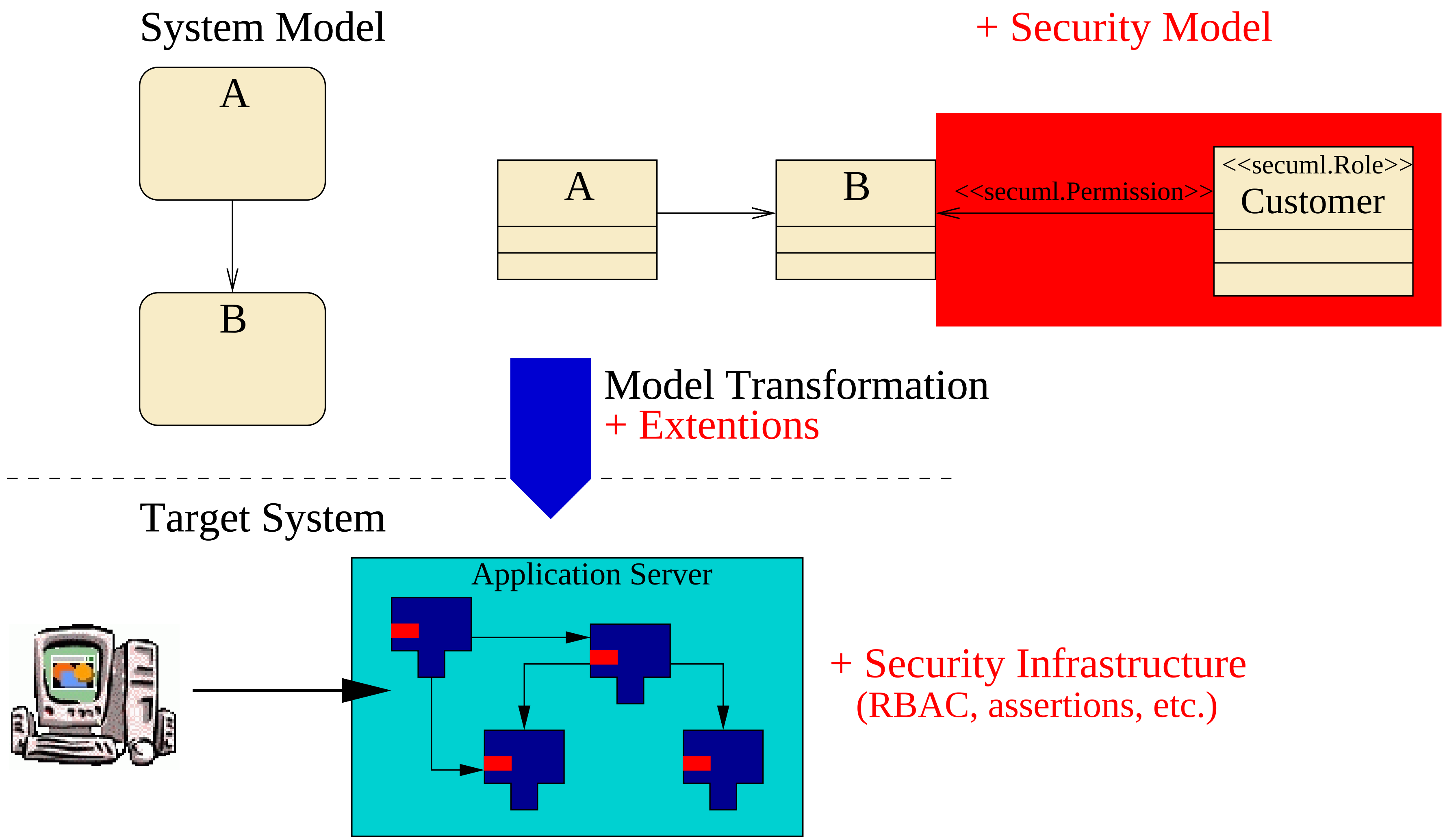
1. All users can create new meetings and read all meeting entries.
2. Only owners may change meeting data, cancel meetings, or delete meeting entries.
3. However, a supervisor can cancel any meeting.

Example — Some Questions

- How do we formalize both kinds of requirements?
- How are requirements refined into multi-tier architectures with support for GUIs, controllers, database back ends ...?
- Can this be done in a way that supports modern standards/technology for modeling (UML), middleware (EJB, .NET, ...), and security?
- How are security infrastructures kept consistent, even when requirements change and evolve, or the underlying technologies themselves change?

We present a methodology & tool addressing these concerns.

Approach: Specialize Model Driven Architecture

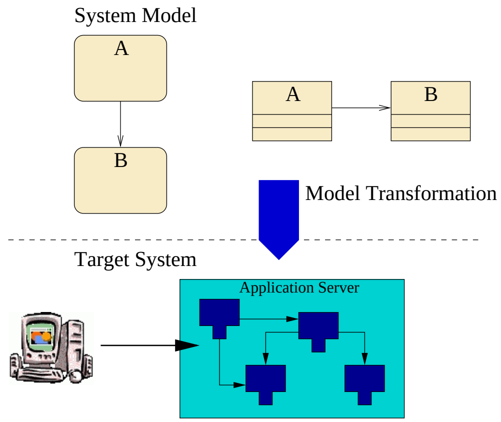


to Model Driven Security.

Requirements Analysis
Security Policies

Implementation
Design Models

Components of MDS



Models:

- Modeling languages combine security and design languages.
- Models specify security and design aspects.

Security Infrastructure: code + standards conform infrastructure.

Assertions, configuration data, calls to interface functions, . . .

Transformation: parameterized by component standard

Examples: J2EE/EJB, .NET, CORBA, ...

Ideas very general.
Approach open with respect to languages and technology.

Road Map

- Motivation and objectives
- ☞ **Background**
- Secure components
- Semantics
- Generating security infrastructures
- Secure controllers
- Experience and conclusions

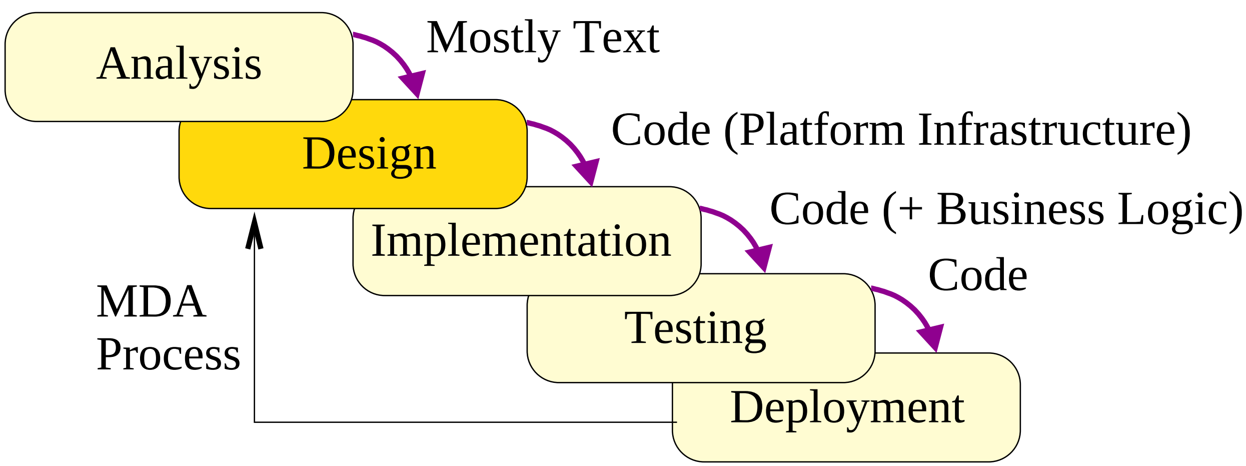
Background

☞ **Model Driven Architecture**

- Unified Modeling Language
- Extensibility and Domain Specific Languages
- Code generation

MDA: the Role of Models

- A model presents a system view useful for conceptual understanding.
- When the models have semantics, they constitute formal specifications and can also be used for (rigorous) analysis and refinement.
- MDA: a model-centric development process



Crucial difference: much of transformation is automated.

MDA: the Role of Standards

- MDA is, e.g., standardized by the Object Management Group.
  - Standards are political, not scientific, constructs.
  - They are valuable for building interoperable tools and for the widespread acceptance of tools and notations used.

- MDA is based on standards for

  Modeling: the Unified Modeling Language, for defining graphical, view-oriented models of requirements and designs.

  Metamodeling: the Meta-Object Facility, for defining modeling languages, like UML.

We will selectively introduce both of these standards.

Background

- Model Driven Architecture

☞ **Unified Modeling Language**

- Extensibility and Domain Specific Languages

- Code generation

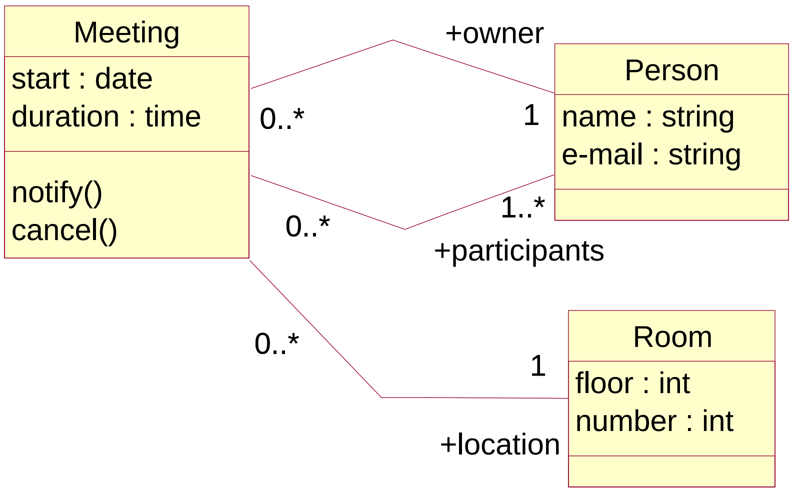
UML

- Family of graphical languages for OO-modeling. Each language:
  - is suitable for formalizing a particular view of systems;
  - has an abstract syntax defining primitives for building models;
  - has a concrete syntax (or notation) for display.
- Also includes the Object Constraint Language.
  - Specification language loosely based on first-order logic.
  - Used to formalize invariants, and pre- and post-conditions.
- A mixed blessing
  - \+ Wide industrial acceptance and considerable tool support.
  - – Semantics just for parts. Not yet a Formal Method.

We focus here on class diagrams and statecharts, presenting the main ideas by example.

Class Diagrams

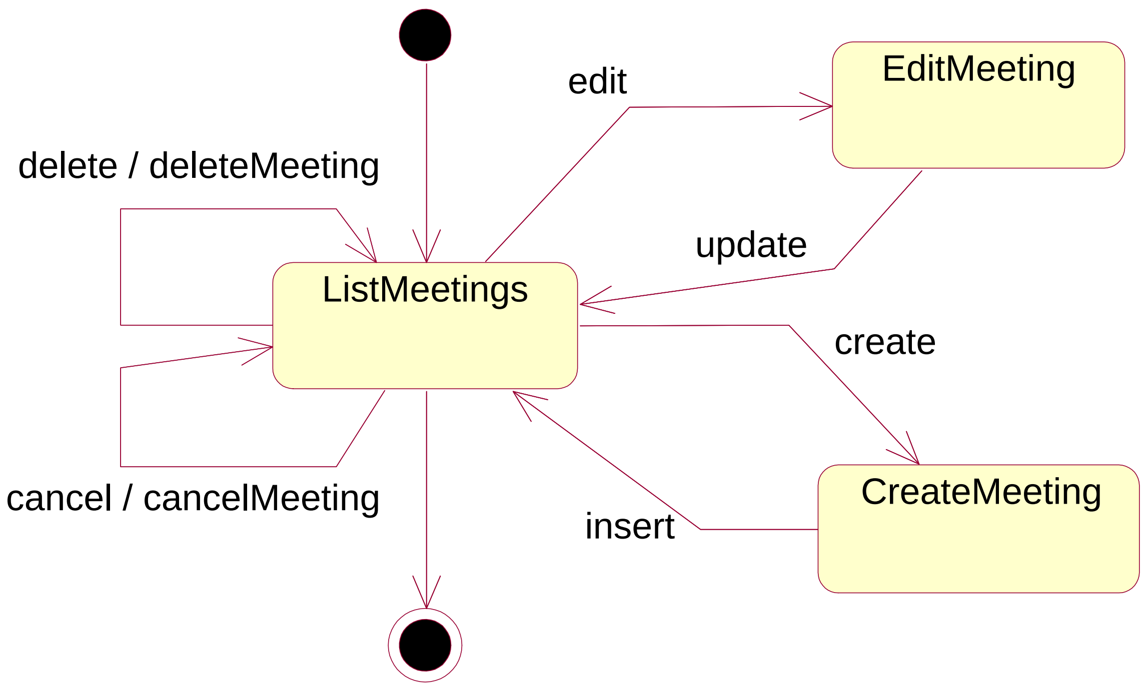
Describe structural aspects of systems. A class formalizes a set of objects with common services, properties, and behaviors. Services are described by methods and properties by attributes and associations.



Sample requirements: The system should manage information about meetings. Each meeting has an owner, a list of participants, a time, and a place. Users may carry out standard operations on meetings such as creating, reading, editing, and deleting them. A user may also cancel a meeting, which deletes the meeting and also notifies all participants by email.

Statecharts

Describes the behavior of a system or class in terms of states and events that cause state transitions.



Sample requirements: Users are presented with a list of meetings. They can perform operations including creating meetings, editing existing meetings, deleting and canceling meetings.

Background

- Model Driven Architecture
- Unified Modeling Language
- ☞ **Extensibility and Domain Specific Languages**
- Code generation

Domain Specific Languages

- UML provides general modeling concepts, yet lacks a vocabulary for modeling Domain Specific Concepts. E.g.,

  Business domains like banking, travel, or health care

  System aspects such as security

- There are various ways to extend UML

  1. by defining new profiles, or
  2. at the level of metamodels.

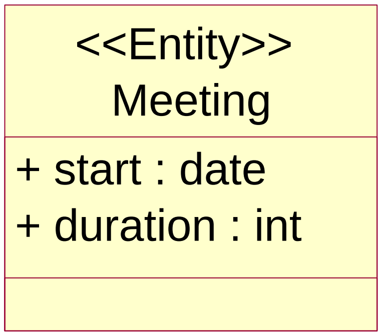
  We will use both of these in our work to define domain specific modeling languages for security and system design.

1) Profiles: Extending Core UML

- UML is defined by a metamodel: core UML.
- Core UML can be extended by defining a UML profile.

  For instance, stereotypes can be declared that introduce modeling primitives by subtyping core UML types and OCL constraints can be used to formalize syntactic well-formedness restrictions.
- Example:

  A class with stereotype <<Entity>> represents a business objects with an associated persistent storage mechanism (e.g., table in a relational database).



- Profiles useful for light-weight specializations.
  Substantial changes use metamodels to define languages directly.

2) Metamodels

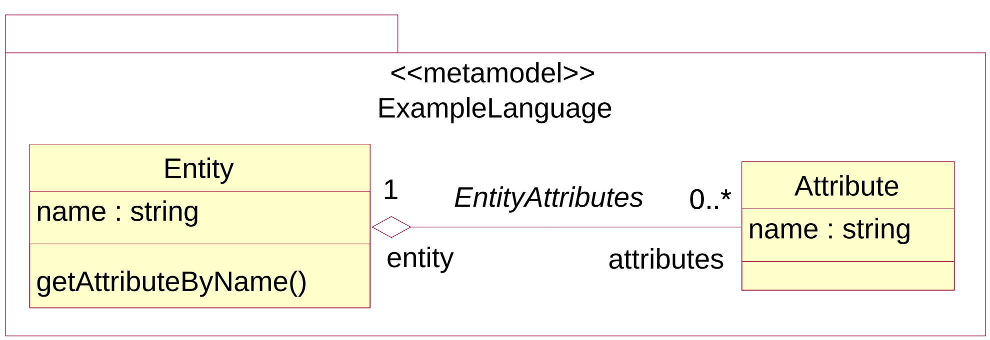
- A metamodel defines the (abstract) syntax of other models.

  Its elements, metaobjects, describe types of model objects.

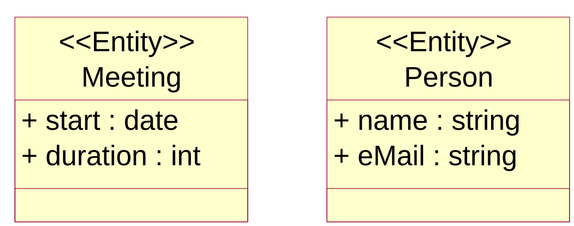
- MOF is a standard for defining metamodels.

Meta level	Description	Example elements
M3	MOF Model	MOF Class, MOF Attribute
M2	Metamodel, defines a language	Entity, Attribute
M1	Model, consisting of instances of M2 elements	Entities "Meeting" and "Person"
M0	Objects and data	Persons "Alice" and "Bob"

M2

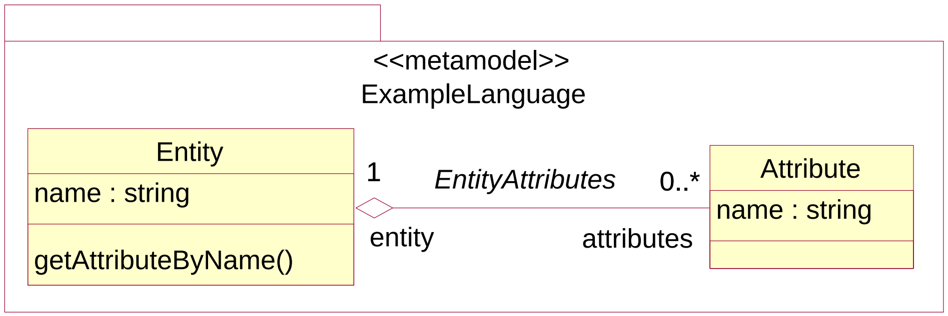


M1

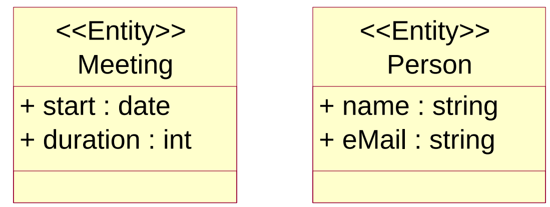


2) Metamodeling (cont.)

M2



M1



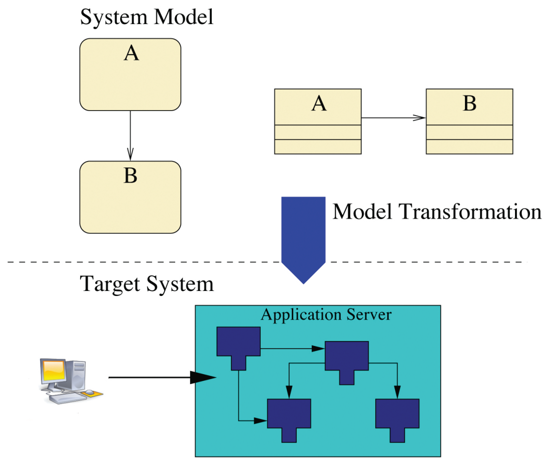
- Abstract syntax of metamodels defined using MOF.
  - Metamodels may be defined using UML notation.
  - Supports OO-metamodels, using concepts like subtyping.
- Concrete syntax of DSL defined by a UML profile.
- MOF/UML tools automatically translate models in concrete syntax into models in abstract syntax for further processing.

Background

- Model Driven Architecture
- Unified Modeling Language
- Extensibility and Domain Specific Languages

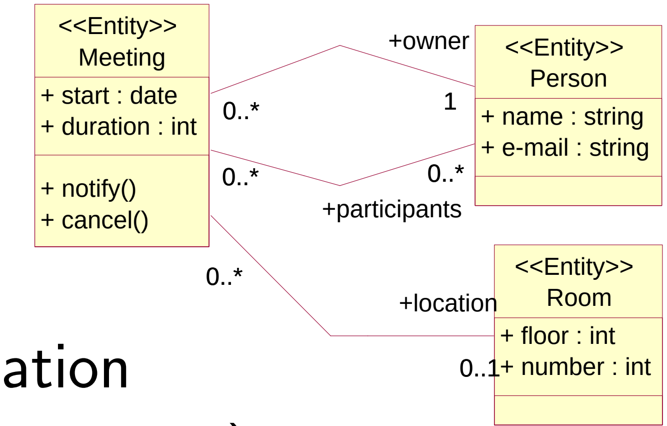
☞ **Code generation**

MDA: Translation



- Fix a platform with a security architecture: J2EE/EJB, .NET, ...
- Consider EJB standard. Beans are:
  1. Server-side components encapsulating application business logic.
  2. Java classes with appropriate structure, interfaces, methods, ...
     + deployment information for installation and configuration.
- Generation rules explain how each kind of model element is translated into part of an EJB system.
- Translation produces Java code and XML deployment descriptors.

MDA Generation by Example



- Entity $\mapsto$ EJB component with implementation class, interfaces (local, remote, home, ...), factory method *create*, finder method *findByPrimaryKey*, ...

- Entity Attribute $\mapsto$ getter/setter methods

```
date getStart() { return start;}
void setStart(date start) { this.start = start; }
```

- Entity Method $\mapsto$ method stub

```
void notify() { }
```

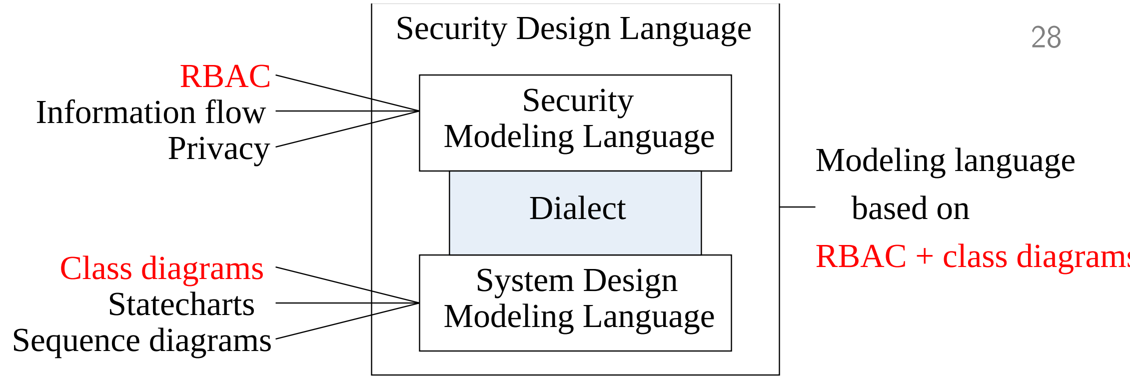
- Association Ends $\mapsto$ schema for maintaining references

```
Collection getParticipants() { return participants; }
void addToParticipants(Person participant)
                                { participants.add(participant); }
void deleteFromParticipants(Person participant)
                                { participants.remove(participant); }
```

Road Map

- Motivation and objectives
- Background
- ☞ **Secure components**
- Semantics
- Generating security infrastructures
- Secure controllers
- Experience and conclusions

Context: Models and Languages



- A Security Design Language glues two languages together.

  Approach open (modulo some semantic requirements).

- Each language is equipped with an abstract and concrete syntax, a semantics, and a technology-dependent translation function.

- Dialect bridges design language with security language by identifying which design elements are protected resources.

- UML employed for

  Metamodeling: Object oriented def. of language syntax (MOF).
  Notation: Concrete language syntax for security design models.

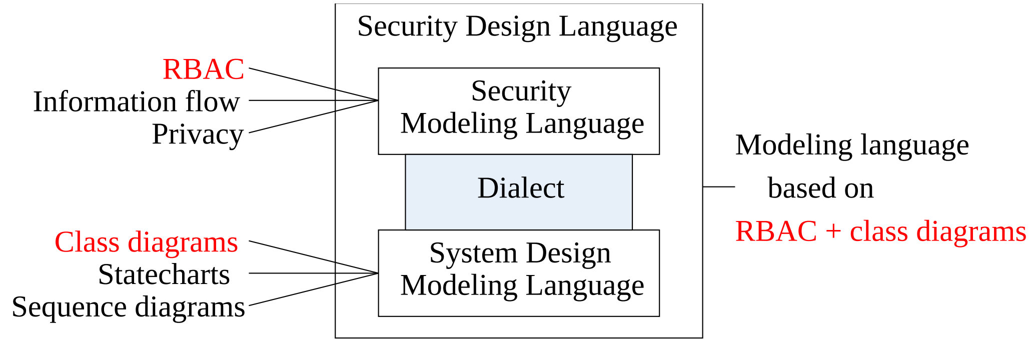
Secure Components

☞ Role-Based Access Control

- Generalization to SecureUML
- Component modeling and combination

We address here relevant concepts and their syntactic representation. Semantics will be handled subsequently.

Security Policies



- Many policies address the confidentiality and integrity of data.

  Confidentiality: No unauthorized access to information
  Integrity: No unauthorized modification of information

- Example: Users may create new meetings and view all meetings, but may only modify the meetings they own.

- Can be formalized as Access Control Policies, specifying which subjects have rights (privileges) to read/write which objects.

- Can be enforced using a reference monitor as protection mechanism.

  Checks whether authenticated users are authorized to perform actions.

- We will focus on access control policies/mechanisms in following.

Access Control

- Two kinds are usually supported.

  Declarative: $u \in$ Users has $p \in$ Permissions $:\Longleftrightarrow (u,p) \in AC$.

  Programmatic: via assertions at relevant program points.
  System environment provides information needed for decision.

- **R**ole **B**ased **A**ccess **C**ontrol is a commonly used declarative model.
  - Roles group privileges.
  - Other additions possible, e.g., hierarchies and sessions.

- These two kinds are often combined, e.g.,

  a user in the role customer may withdraw money from an account
  when he is the owner and the amount is less than 1,000 SFr.

Access Control — Declarative

- Declaratively: authorization is specified by a relation.

  A user is granted access iff he has the required permission.

$$u \in \text{Users has } p \in \text{Permissions} :\Longleftrightarrow (u, p) \in \text{AC}.$$

- Example:

User
Alice
Bob
John

User	Permission
Alice	read file a
Alice	write file a
Alice	start application x
Alice	start application y
Bob	read file a
Bob	write file a
Bob	start application x
John	read file a
John	write file a
John	start application x

Permission
read file a
write file a
start application x
start application y

Role-Based Access Control

- Role-Based Access Control decouples users and permissions by roles, representing jobs or functions.

- Formalized by a set Roles and the relations UA $\subseteq$ Users $\times$ Roles and PA $\subseteq$ Roles $\times$ Permissions, where

$$\mathsf{AC} := \mathsf{PA} \circ \mathsf{UA}$$

i.e.,
$$\begin{aligned} \mathsf{AC} \;:=\; & \{(u,p) \in \mathsf{Users} \times \mathsf{Permissions} \mid \\ & \exists r \in \mathsf{Roles} : (u,r) \in \mathsf{UA} \wedge (r,p) \in \mathsf{PA}\} \;. \end{aligned}$$

- Example:

User	Role
Alice	User
Alice	Superuser
Bob	User
John	User

Role
User
Superuser

Role	Permission
User	read file a
User	write file a
User	start application x
Superuser	start application y

Result is increased abstraction and more manageable policies.

RBAC — Extensions

1. Role hierarchy (for $\geq$ a partial order):

$$\begin{tikzcd} \text{Superuser} \arrow[d] \\ \text{User} \end{tikzcd}$$

$$\mathsf{AC} := \mathsf{PA} \circ \geq \circ \mathsf{UA}$$

   Larger roles inherit permissions from all smaller roles

2. Hierarchies on users (UA) and permissions (PA).

3. Authorization constraints: formulae used to make stateful access control decisions.

   Example: a user in the role customer may withdraw money from an account when he is the owner and the amount is less than 1,000 SFr.

Secure Components

- Role-Based Access Control

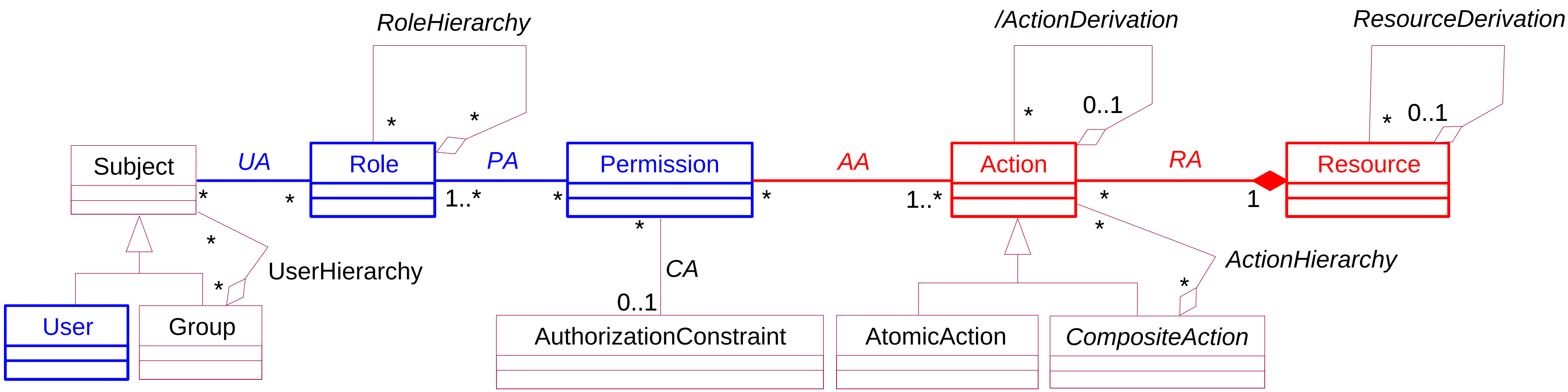
☞ **Generalization to SecureUML**

- Component modeling and combination

SecureUML – Syntax

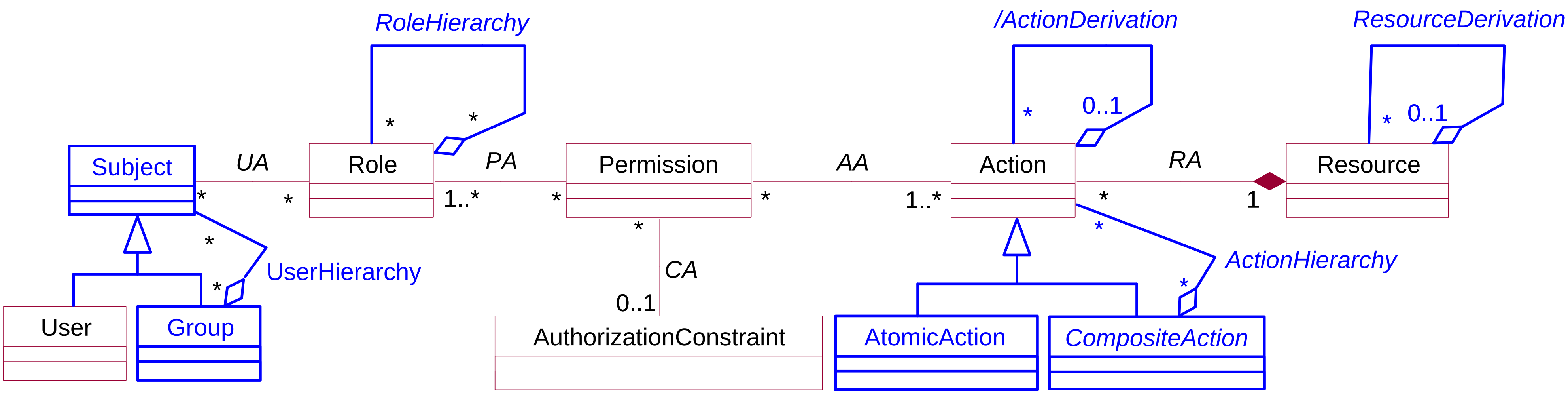
- Abstract syntax defined by a MOF metamodel.
- Concrete syntax based on UML and defined with a UML profile.
- Syntax and semantics based on an extension of RBAC.
- **Key idea**
  - An access control policy formalizes the permissions to perform actions on (protected) resources.
  - We leave these open as types whose elements are not fixed.
  - Elements specified during combination with design language (via subtyping from existing types).

Users, Roles and Typed Permissions



- Left hand part: essentially Standard RBAC
- Right hand part: permissions are factored into the ability to carry out actions on resources.
  - Resource is the base class of all model elements representing protected resources (e.g. “Class”, “State”, ’Action”).
  - Actions of a “Class” could be “Create”, “Read”, “Delete” ...

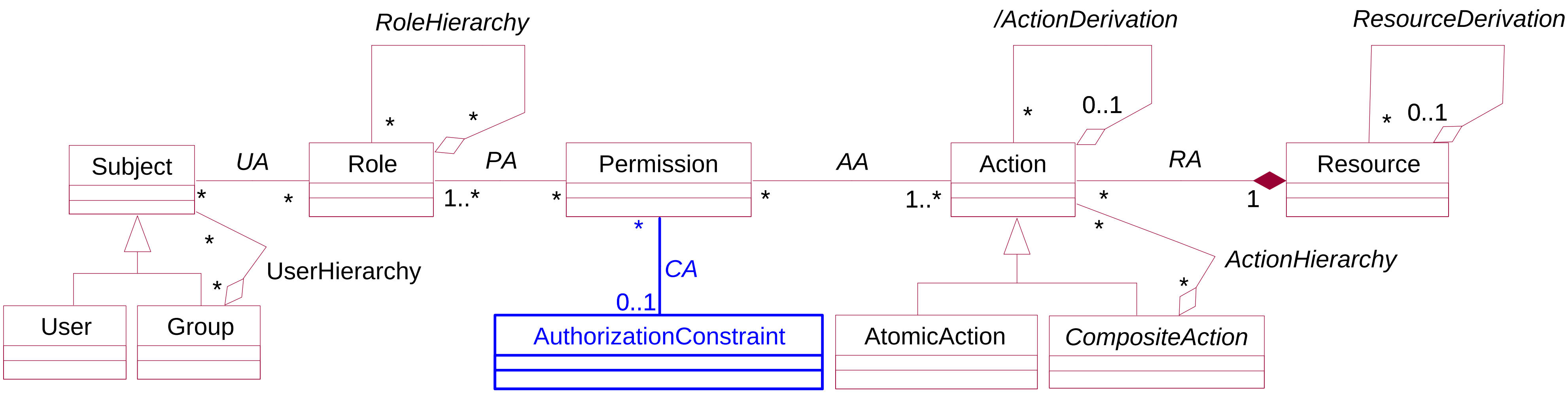
Hierarchies over Users, Roles and Actions



- UserHierarchy: Users (and groups) are organized in groups.
- RoleHierarchy: Roles can be in an inheritance hierarchy.
- ActionHierarchy: E.g., “FullAccess” is a super-action of “Read”.
- ActionDerivation/ResourceDerivation: Details technical & omitted.

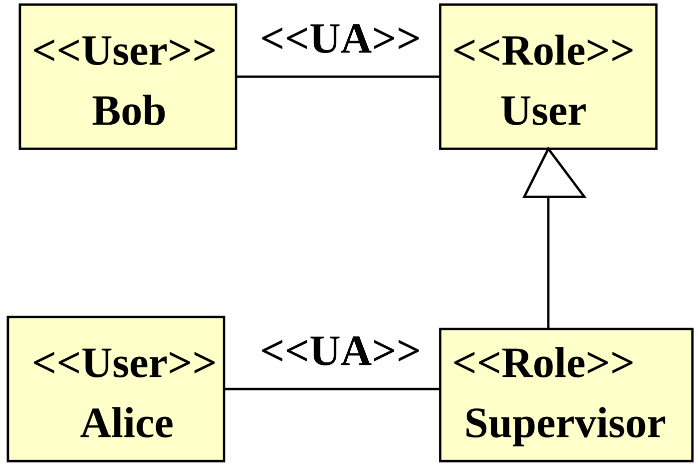
Note: hierarchies modeled using UML-aggregation associations.

Authorization Constraints



- A permission can be restricted by an authorization constraint. E.g., user is account owner and amount is less than 1,000 CHF.
- This assertion describes an additional condition that must hold in order to grant access. Condition on:
  - the state of the resources of the assigned actions,
  - properties of method arguments (name of the calling user), or
  - global system properties (time, date)

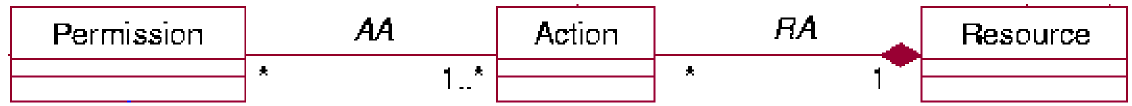
Roles and Users



- Users, Roles, and Groups (here none) defined by stereotyped classes.
- Hierarchies defined using inheritance.
- Relations defined using steroretyped associations.

NOTE: User administration is not a design-time issue and hence usually not part of a design model. In practice, these assignments are made by system administrators after system deployment.

Permissions



- Modeling permissions require that actions and resources have already been defined.

  Possible only possibly after language combination. (Coming up!)

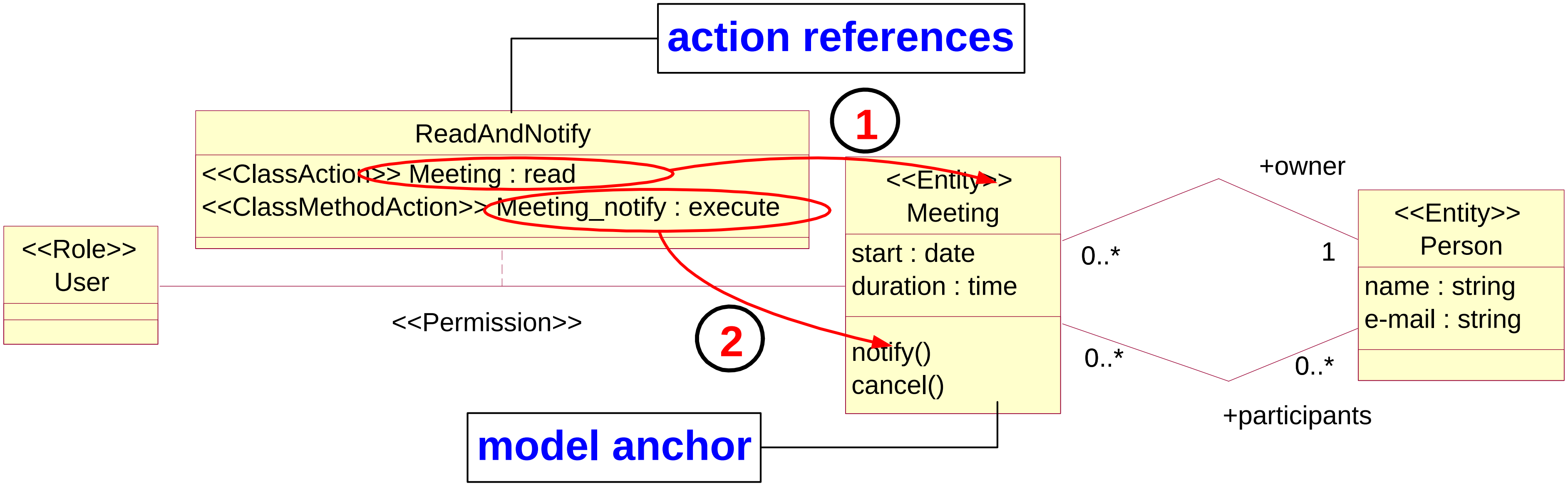
- A permission binds one or more actions to a single resource.

- Concrete syntax could directly reflect abstract syntax

  Specify two relations: Permission $\Leftrightarrow$ Action and Action $\Leftrightarrow$ Resource.

- Alternative: use association class to specify one ternary relation.
  - Attributes of association relate permissions with actions.
  - Actions identified by resource name and action name.

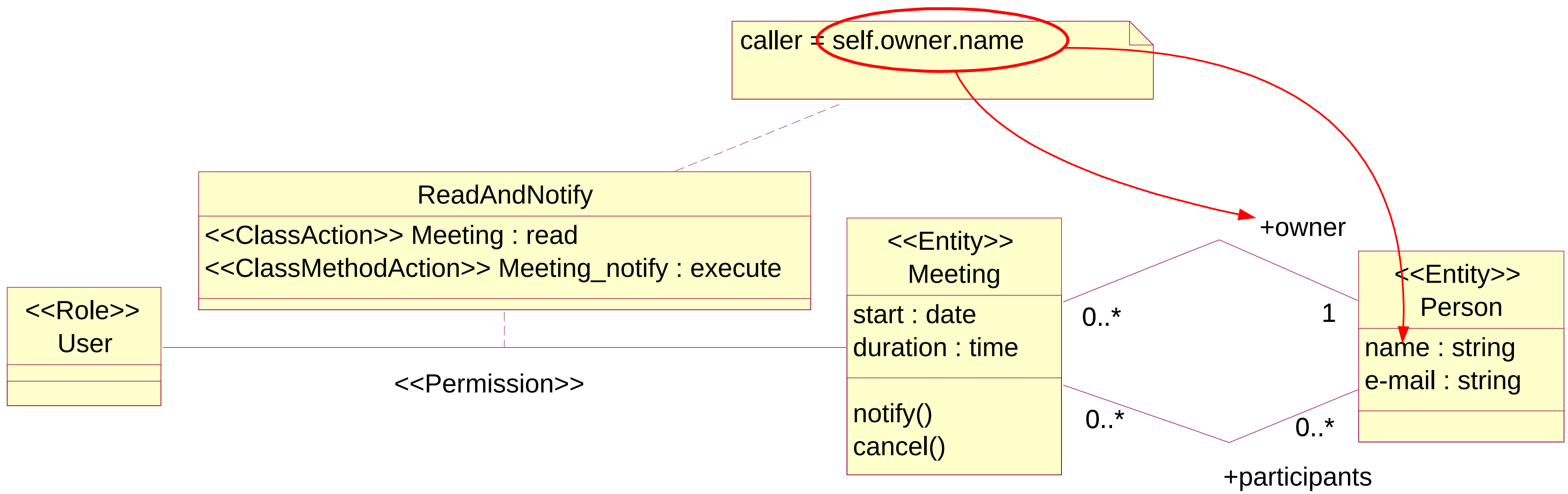
Permissions (Cont.)



- Represented as an association class connecting a role and a class (model anchor).
- Permission (action references) may assign actions to (1) the model anchor or (2) its sub-elements.

  E.g., the first action says that users have permission to read meetings. We will see this means they may execute all side-effect free methods and access all attribute ends of meetings.

Authorization Constraints



- Expressions are given in an OCL subset
  - constant symbols: *self* and *caller* (authenticated name of caller)
  - attributes and side-effect free methods
  - navigation expressions (association ends)
  - Logical (and, or, not) and relational ($=, >, <, <>$) operators
  - Existentially quantified expressions
- Example: "`caller = self.owner.name`"

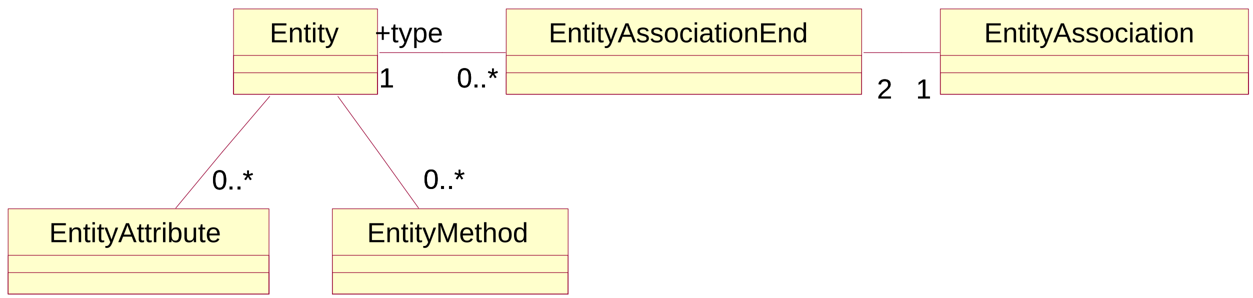
Secure Components

- Role-Based Access Control
- Generalization to SecureUML

☞ **Component modeling and combination**

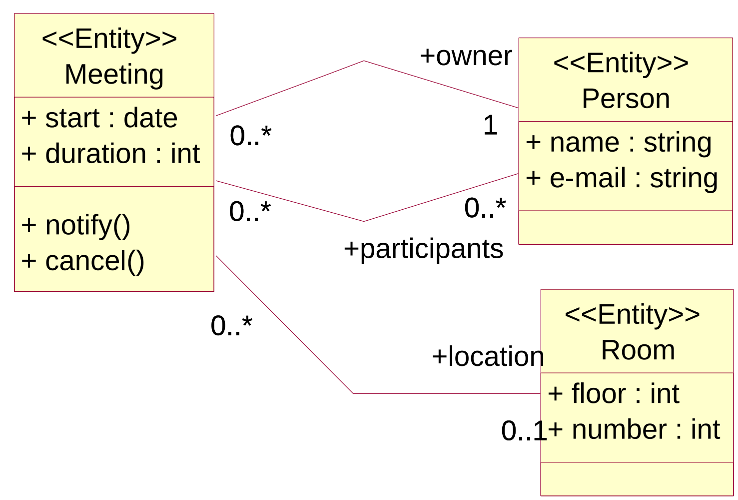
A Design Modeling Language for Components

- ComponentUML: a class-based language for data modeling.

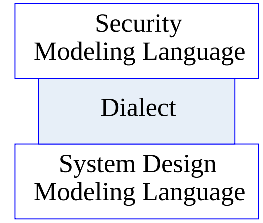


- Example design: group meeting administration system.

Each meeting has an owner, participants, a time, and possibly a location. Users carry out operations on meetings like create, read, edit, delete, or cancel (which notifies the participants).



Combination with SecureUML



1. Combine syntax of both modeling languages

   Merge abstract syntax by importing SecureUML metamodel into metamodel of ComponentUML.

   Merge notation and **define well-formedness rules** in OCL. E.g., restrict permissions to those cases with stereotype «Entity».

2. Identify protected resources

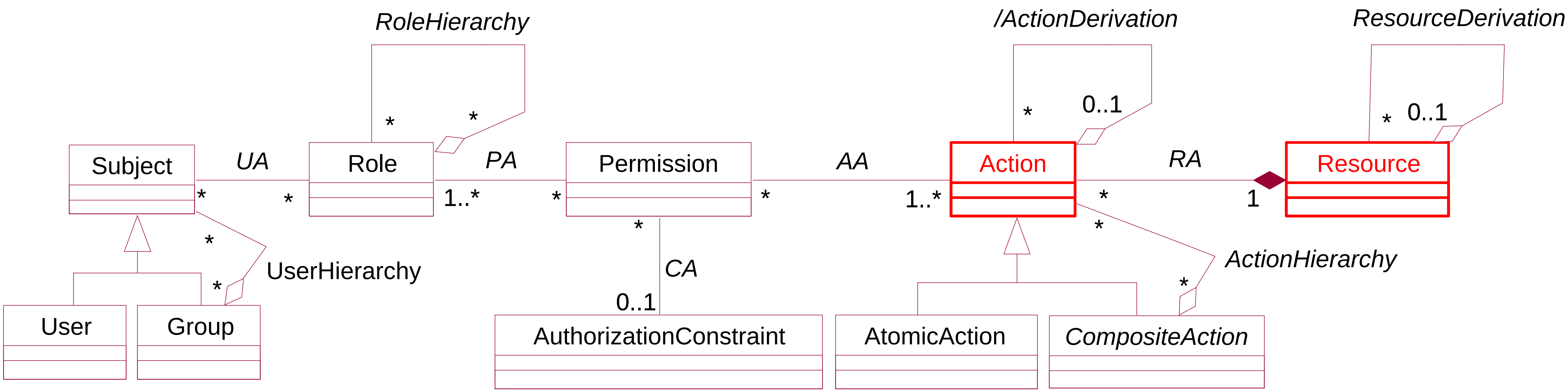
3. Identify resource actions

4. Define action hierarchy

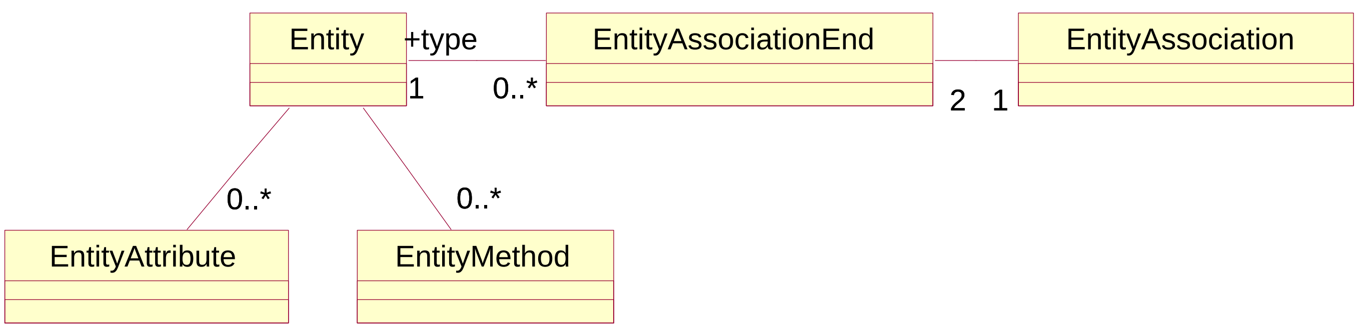
First task is (mostly) automated. Remainder are creative tasks. They constitute what we have called a dialect or glue.

Defining a Dialect

Security Modeling Language = SecureUML

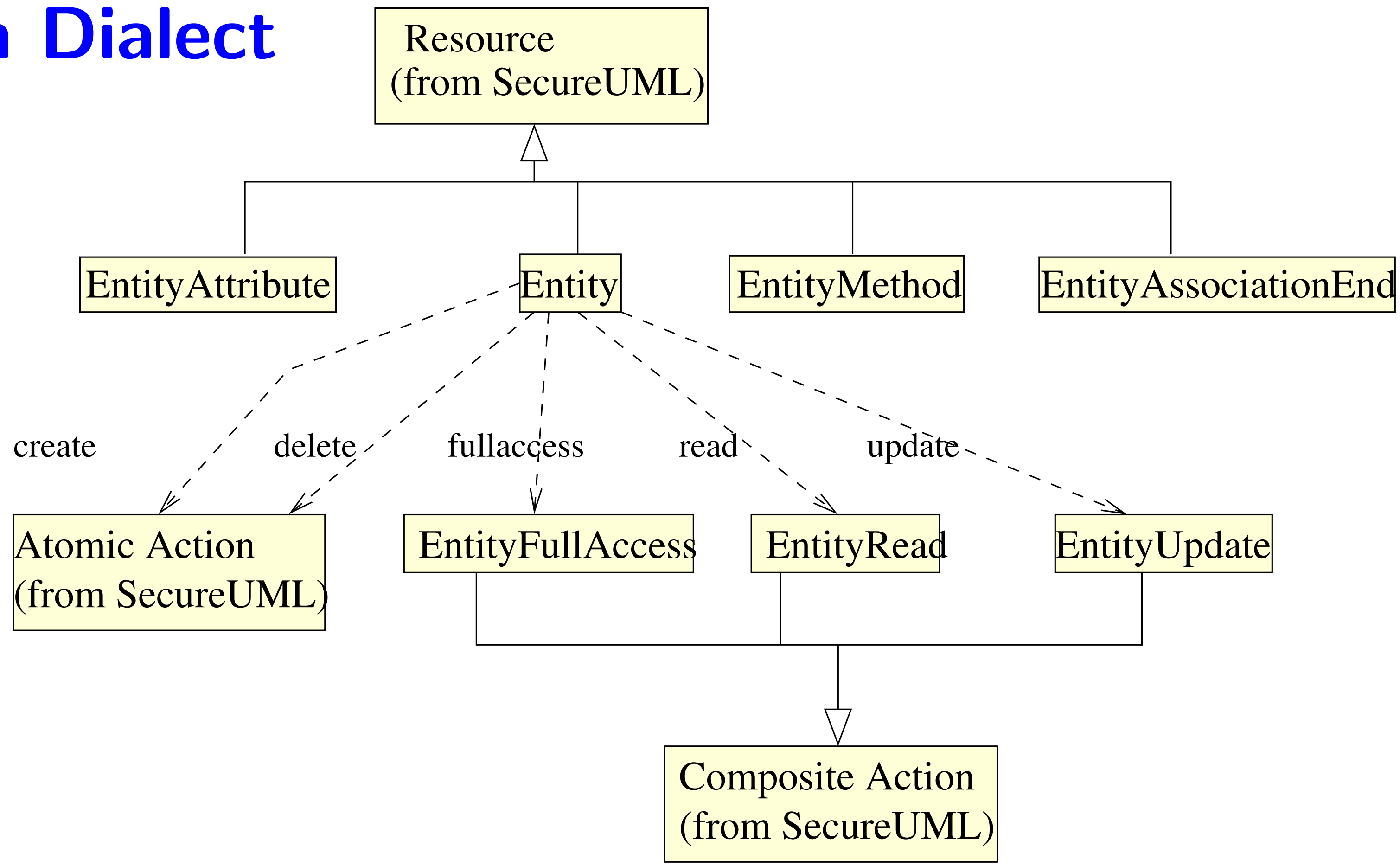


System Design Modeling Language = Component UML



What are the resources and actions of ComponentUML?

Defining a Dialect



- Resources identified using subtyping.
- Resource actions defined using named dependencies from resource types to action classes (either Atomic Action or a subtype of Composite Action).

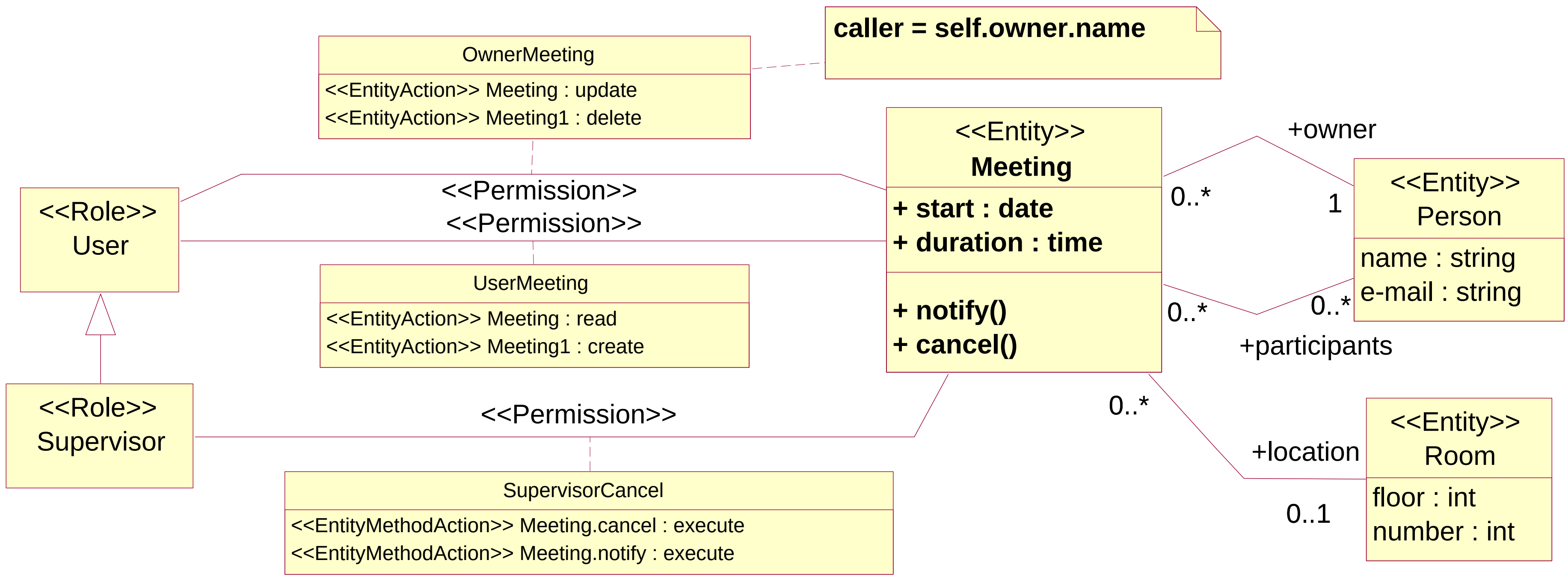
Action Hierarchy

resource type	action	subordinated actions (with blue atomic actions)
Entity	full access	*create*, *read*, *update* and *delete* of the entity
Entity	read	*read* for all attributes and association ends of the entity *execute* for all side-effect free methods of the entity
Entity	update	*update* for all attributes of the entity *add* and *delete* all association ends of the entity *execute* for all methods with side-effects of the entity
Attribute	full access	*read* and *update* of the attribute
Association End	full access	*read*, *add* and *delete* of the association end

OCL formulae used to formalize hierarchy. E.g., following states that the composite action *EntityFullAccess* is larger than the actions *create*, *read*, *update*, and *delete* of the entity the action belongs to.

context EntityFullAccess inv:
  subordinatedActions = resource.actions->select(
    name="create" or name="read" or name="update" or name="delete")

Modeling a Security Policy



1. All users can create new meetings and read all meeting entries.
2. Only owners may change meeting data or delete meeting entries.
3. However, a supervisor can cancel any meeting.

Road Map

- Motivation and objectives
- Background
- Secure components

☞ **Semantics**

What do all these boxes and arrows actually mean?
Here we provide just a **sketch**. Full details provided in TOSEM paper.

- Generating security infrastructures
- Secure controllers
- Experience and conclusions

SecureUML formalizes two kinds of AC decisions

Declarative Access Control where decisions depend on **static information**:
the assignments of users u and permissions (to actions a) to roles.

AC decision formalized by $\mathfrak{S}_{RBAC} \models \phi_{RBAC}(u, a)$

Programmatic Access Control where decisions depend on **dynamic information**:
the satisfaction of authorization constraints in the current system state.

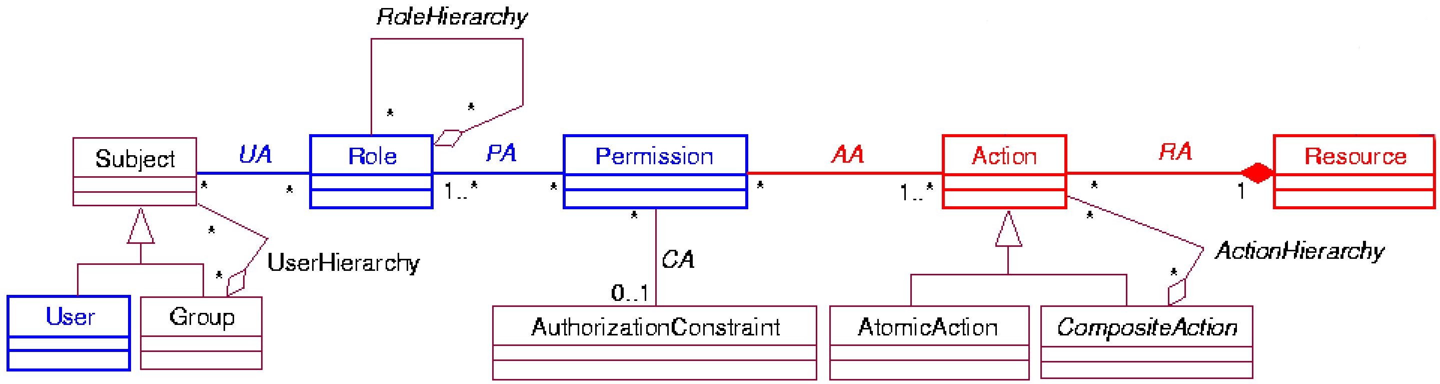
AC decision formalized as $\mathfrak{S}_{St} \models \phi^{p}_{st}$

Where

- $\mathfrak{S}_{RBAC}$ is a first-order structure formalizing the static (RBAC) information
- $\phi_{RBAC}(u, a)$ is a first-order formula formalizing that user u can perform action a
- $\mathfrak{S}_{St}$ is a first-order structure formalizing the system state
- ϕ^{p}_{st} is a first-order formula formalizing restriction on permission p

Decisions are combined. Roughly $\langle \mathfrak{S}_{RBAC}, \mathfrak{S}_{St} \rangle \models \phi_{AC}(u, a)$, where ϕ_{AC} states that u has permission to execute action a and the associated authorization constraint holds.

Declarative Semantics



- Order-sorted signature $\Sigma_{RBAC} = (\mathcal{S}_{RBAC}, \mathcal{F}_{RBAC}, \mathcal{P}_{RBAC})$.

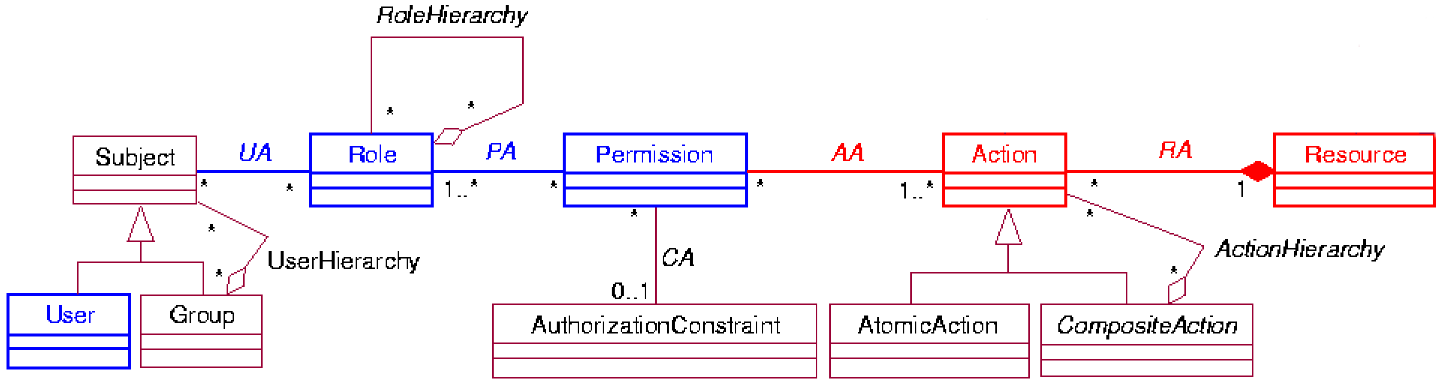
$$\mathcal{S}_{RBAC} = \{\textit{Users}, \textit{Subjects}, \textit{Roles}, \textit{Permissions}, \textit{Actions}\} \;\; ,$$

$$\mathcal{F}_{RBAC} = \emptyset \;\; ,$$

$$\mathcal{P}_{RBAC} = \{\geq_{\textit{Subjects}}, UA, \geq_{\textit{Roles}}, PA, AA, \geq_{\textit{Actions}}\} \;\; ,$$

- *Users* is a subsort of *Subjects*.
- Types as expected, e.g., UA has type *Subjects* $\times$ *Roles*.
- UA, PA, and AA correspond to identically named associations in metamodel.
- $\geq_{\textit{Subjects}}$, $\geq_{\textit{Roles}}$, and $\geq_{\textit{Actions}}$ name hierarchies on users, roles, and actions.

Declarative Semantics (cont.)

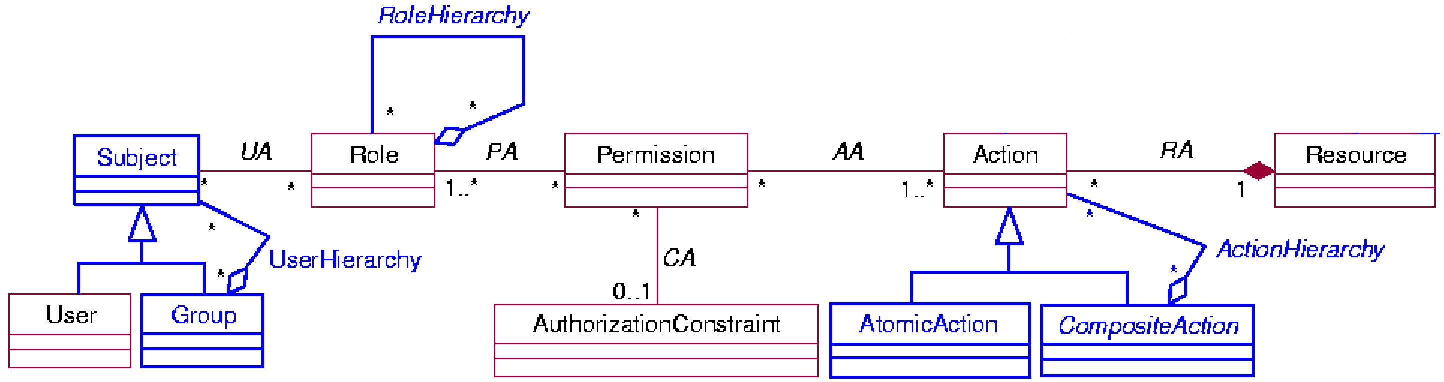


- A SecureUML model straightforwardly defines a Σ_{RBAC}-structure $\mathfrak{S}_{St}$.
  - Users (Roles, ...) in model $\mapsto$ elements of set Users (Roles ...).
  - Associations (e.g., between users & roles) $\mapsto$ tuples in associated relations (e.g., UA).
- $\phi_{RBAC}(u, a)$ formalizes standard RBAC semantics (here without hierarchies)
  - "Can user u perform permission p?"
    $\phi_{RBAC}(u, p) \iff (u, p) \in \mathsf{AC}$, where $\mathsf{AC} := \mathsf{PA} \circ \mathsf{UA}$.
  - is refined to: "Does user u have the permission to carry out action a?"
    $\phi_{RBAC}(u, a) \iff (u, a) \in \mathsf{AC}$, where $\mathsf{AC} := \mathsf{AA} \circ \mathsf{PA} \circ \mathsf{UA}$, i.e.
  - In first-order logic:

$$\phi_{RBAC}(u, a) \iff \exists r, p : \mathsf{UA}(u, r) \wedge \mathsf{PA}(r, p) \wedge \mathsf{AA}(p, a)\}$$

- AC Decision Problem is: $\mathfrak{S}_{RBAC} \models \phi_{RBAC}(u, a)$.

Adding Hierarchies



- Additional ordering relations $\geq_{Subjects}$, $\geq_{Roles}$, and $\geq_{Actions}$:
  - $\geq_{Subjects}$ interpreted by reflexive, transitive closure of $UserHierarchy$, where a group is larger than all its contained subjects.
  - $\geq_{Roles}$ and $\geq_{Actions}$ are interpreted analogously using $RoleHierarchy$ and $ActionHierarchy$.
- ϕ_{RBAC} now formalizes $\geq_{Actions} \circ \mathsf{AA} \circ \mathsf{PA} \circ \geq_{Roles} \circ \mathsf{UA} \circ \leq_{Subjects}$

$$\text{i.e., } \phi_{RBAC}(u,a) \quad = \quad \exists s \in \mathit{Subjects}, r_1, r_2 \in \mathit{Roles}, p \in \mathit{Permissions}, a' \in \mathit{Actions}.$$
$$u \leq_{\mathit{Subjects}} s \wedge UA(s, r_1) \wedge r_1 \geq_{\mathit{Roles}} r_2 \wedge$$
$$PA(r_2, p) \wedge AA(p, a') \wedge a' \geq_{\mathit{Actions}} a \ ,$$

Authorization Constraints

- Authorization constraints are OCL formulae, attached to permissions.

  business hours: time.hour >= 8 and time.hour <= 17
  caller is owner: caller = self.owner.name

- Straightforward translation into sorted FOL, e.g.,

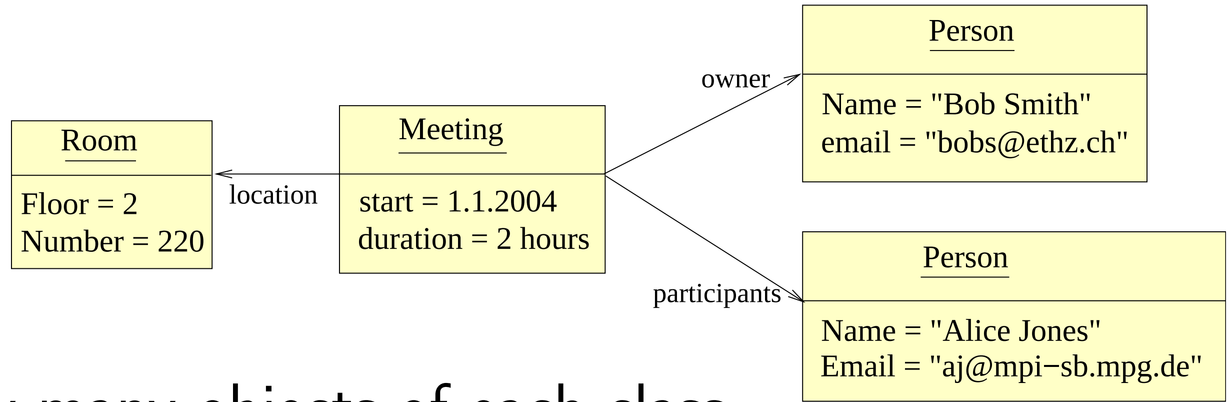
$$hour(time) \geq 8 \wedge hour(time) \leq 17$$
$$caller = name(owner(self))$$

- The signature Σ_{St}. for constraints is determined by the design modeling language

  $\mathcal{S}_{St}$**:** sort for each class in the system model
  $\mathcal{F}_{St}$**:** function symbol for each attribute, side-effect free method, and n-1 association.
  $\mathcal{P}_{St}$**:** predicate symbol for each m-n association.

Constraint Semantics

- A system snapshot during execution defines a state

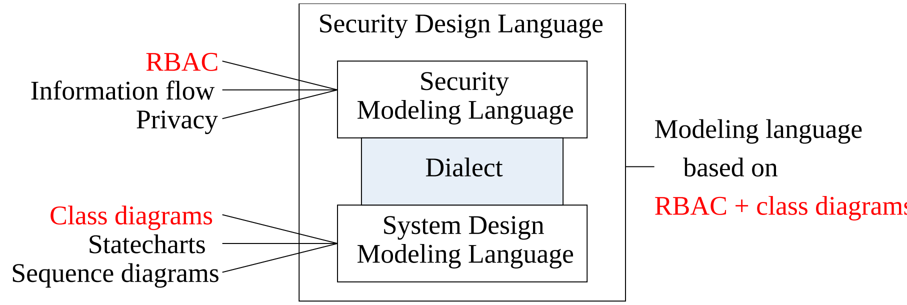


- In general, there are finitely many objects of each class C, each with its own attribute values and references to other objects.

- Interpretation idea
  - ▶ Each sort interpreted by a finite set of “objects”.
  - ▶ Attributes and references define functions (or relations) from objects to corresponding values.
  - ▶ Currently executing object of class C gives interpretation for self_C.

- A constraint ϕ_{St} is satisfied iff $\mathfrak{S}_{St} \models \phi_{St}$.

Semantic of Combinations



- SecureUML semantics has a fixed static part plus a stateful part, dependent on the notion of state defined by design modeling language.

- What is the combination's semantics?

  Intuitively: system with access control should behave as before, except that certain actions are disallowed in certain states.

  Formally: semantics defined in terms of labeled transition systems.

- Minimal assumptions required on semantics of design language:

  Semantics must be expressible as an LTS, whose states provide a structure for interpreting OCL assertions.

Semantic Requirements of Design Language

Semantics definable as a LTS $\Delta = (Q, A, \delta)$

– set Q of nodes consists of Σ_{St}-structures

– edges are labeled with elements from a set of actions A

– $\delta \subseteq Q \times A \times Q$ is transition relation

System behavior defined by traces as is standard:

$s_0 \stackrel{a_0}{\to} s_1 \stackrel{a_1}{\to} \ldots$ is a trace iff $(s_i, a_i, s_{i+1}) \in \delta$, for all i, $0 \leq i$.

Combination with SecureUML

- Combining Δ with SecureUML yields LTS $\Delta_{AC} = (Q_{AC}, A_{AC}, \delta_{AC})$.

  - $Q_{AC} = Q_{RBAC} \times Q$, combines system states with RBAC

    Here Q_{RBAC} denotes universe of all finite Σ_{RBAC}-structures.

  - $A_{AC} = A$ is unchanged.

  - Transition function defined by

$$\begin{aligned}\delta_{AC} \;=\; & \{((q_{RBAC}, q), a, (q_{RBAC}, q')) \mid \\ & \quad (q, a, q') \in \delta \wedge \langle q_{RBAC}, q\rangle \models \phi_{AC}(u, a)\}\end{aligned}$$

- In δ_{AC}, just those traces with prohibited actions are removed.

- This account is both general and independent of UML.

Example: SecureUML + ComponentUML

- ComponentUML as an LTS $\Delta = (Q, A, \delta)$
  - Q is the universe of all possible system states: interpretations over the signature Σ_{St} with finitely many objects for each entity.
  - Family of actions A defined by methods and their parameters. E.g., the action (set_{at}, e, v) denotes setting the attribute at of entity e to value v.
  - δ defined by (transition-system) semantics of methods themselves. E.g., above setter action would lead to a new state where only the term representing e is changed to reflect the update of a with v.
- Combined semantics $\Delta_{AC} = (Q_{AC}, A_{AC}, \delta_{AC})$ as just described.

Road Map

- Motivation and objectives
- Background
- Secure components
- Semantics
- ☞ **Generating security infrastructures**
- Secure controllers
- Experience and conclusions

Generating Security Infrastructures

☞ **Generating EJB Infrastructures.**

- Motivation
- Basics of EJB and EJB access control
- Generation rules

- Generating .NET infrastructures.

Why Transform?

Decreases burden on programmer.

Faster adaption to changing requirements.

Scales better when porting to different platforms.

Correctness of generation can be proved, once and for all.

☞ enables a faster, cheaper, and more secure development process.

Let's look at this first for Enterprise Java Beans (EJBs), a widely used component architecture.

EJB: Declarative AC

```
<method-permission>
  <role-name>Supervisor</role-name>
  <method>
    <ejb-name>Meeting</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>cancel</method-name>
    <method-params/>
  </method>
</method-permission>
```

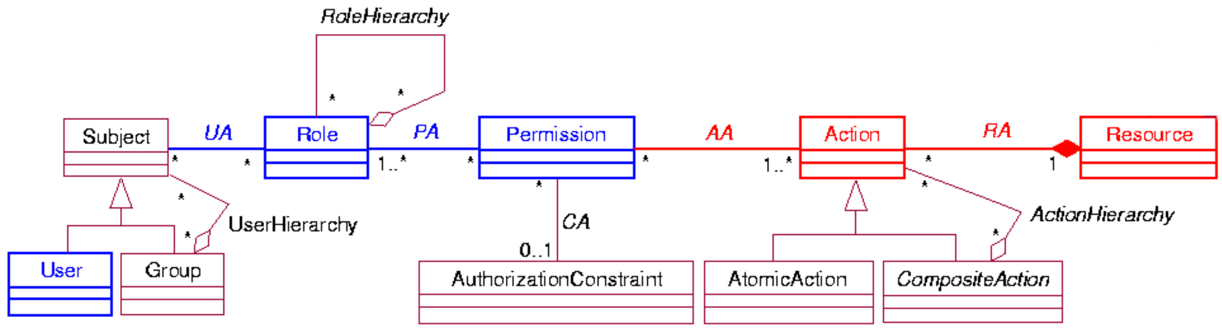
- Deployment descriptors record information for declarative AC.
- EJB supports only vanilla RBAC without hierarchies, where protected resources are individual methods.

EJB: Programmatic AC

```
if( !(ctxt.isCallerInRole("SuperVisor")
      || ctxt.getCallerPrincipal.getName().equals(
        getOwner.getName()))){
throw new AccessControlException("Access Denied");
}
```

Assertions use programmatic access control support of EJB Server to access security-relevant data of current user, e.g., his name or his roles.

Transformation Rules RBAC



For each atomic action a:

- determine the corresponding EJB method(s) m.
- compute the set of Roles R that have access to the action a:

$$R := \{r \in \text{Roles} \mid (r, a) \in \geq_{\text{Actions}} \circ \text{AA} \circ \text{PA} \circ \geq_{\text{Roles}}\} \ .$$

☞ generate the deployment-descriptor code (with $R = \{r_1, \ldots, r_n\}$):

```
<method-permission>
  <security-role>r1</security-role>
  ...
  <security-role>rn</security-role>
  <method>m</method>
</method-permission>
```

Transformation Rules: Assertions

For each atomic action a on a method m:

- compute the set of permissions P for this action:

$$P := \{p \in \text{Permissions} \mid (p, a) \in \geq_{\text{Actions}} \circ \text{AA}\}$$

- for each $p \in P$, compute the set of roles $R(p)$ assigned to the permission p:

$$R(p) := \{r \in \text{Roles} \mid (r, p) \in \text{PA} \circ \geq_{\text{Roles}}\}$$

- Check, if one of the $p \in P$ has an authorization constraint attached.

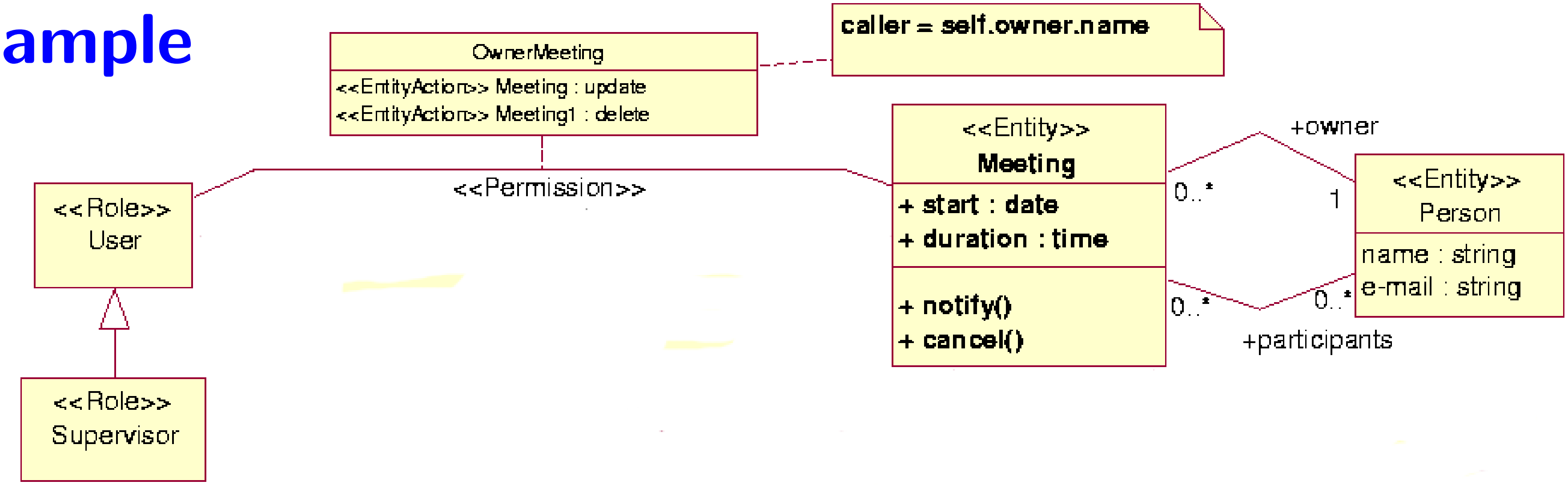
☞ if yes, include at the start of the method m the assertion:

$$\texttt{if (!(}\bigvee_{p \in P}\Big(\Big(\bigvee_{r \in \mathsf{R}(p)} \texttt{ctxt.isCallerInRole(}r\texttt{)}\Big) \wedge \text{Constraint}(p)\Big)\texttt{))}$$

$$\texttt{throw new AccessControlException("Access denied.");}\ ,$$

where Constraint(p) is attached constraint (or *true*) in Java syntax.

Example

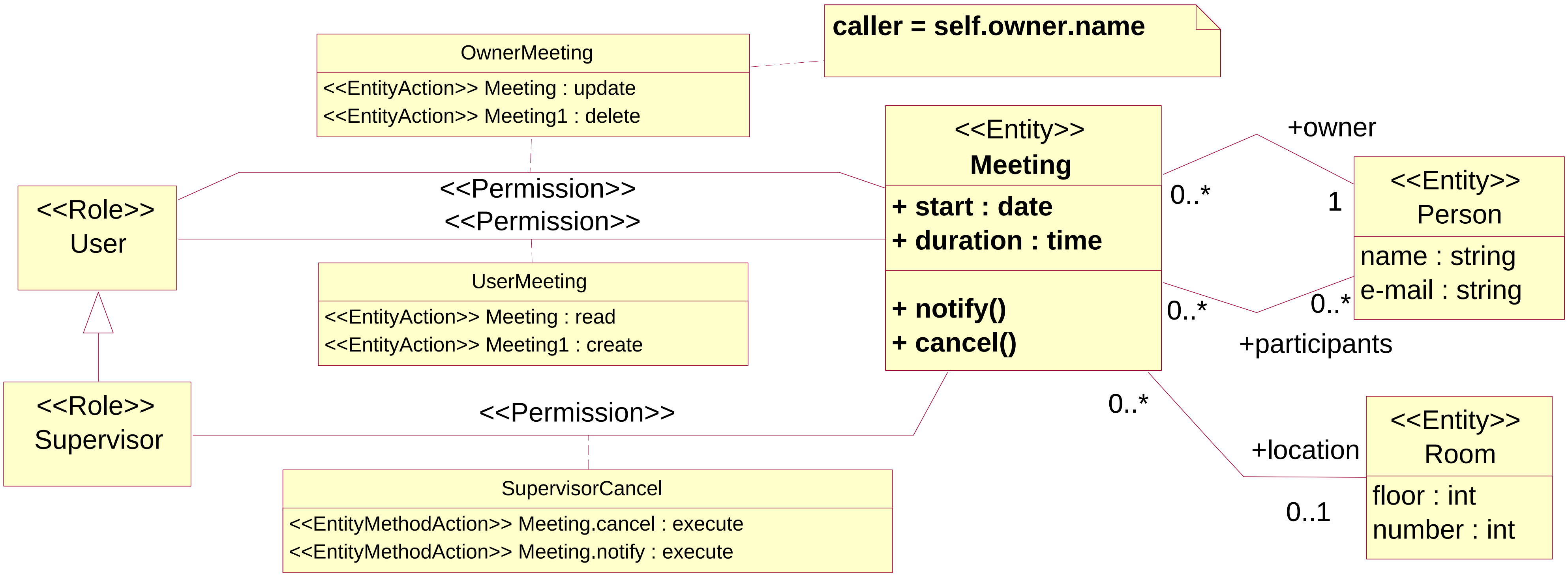


generates both RBAC configuration data and Java code:

```
<method-permission>
  <role-name>User</role-name>
  <role-name>Supervisor</role-name>
  <method>
    <ejb-name>Meeting</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>setStart<//method-name>
</method>
</method-permission>
```

```
public void setStart(Date start)
{
if (!((ctxt.isCallerInRole("User") ||
     ctxt.isCallerInRole("Supervisor"))
   && ctxt.getCallerPrincipal.getName().equals(
                getOwner().getName())))
 )) throw new AccessControlException("Access
denied.");
...
}
```

Overall Model



Generates 179 lines of XML and 268 lines of Java.
Which would you rather maintain or port?

Generating Security Infrastructures

- Generating EJB infrastructures.

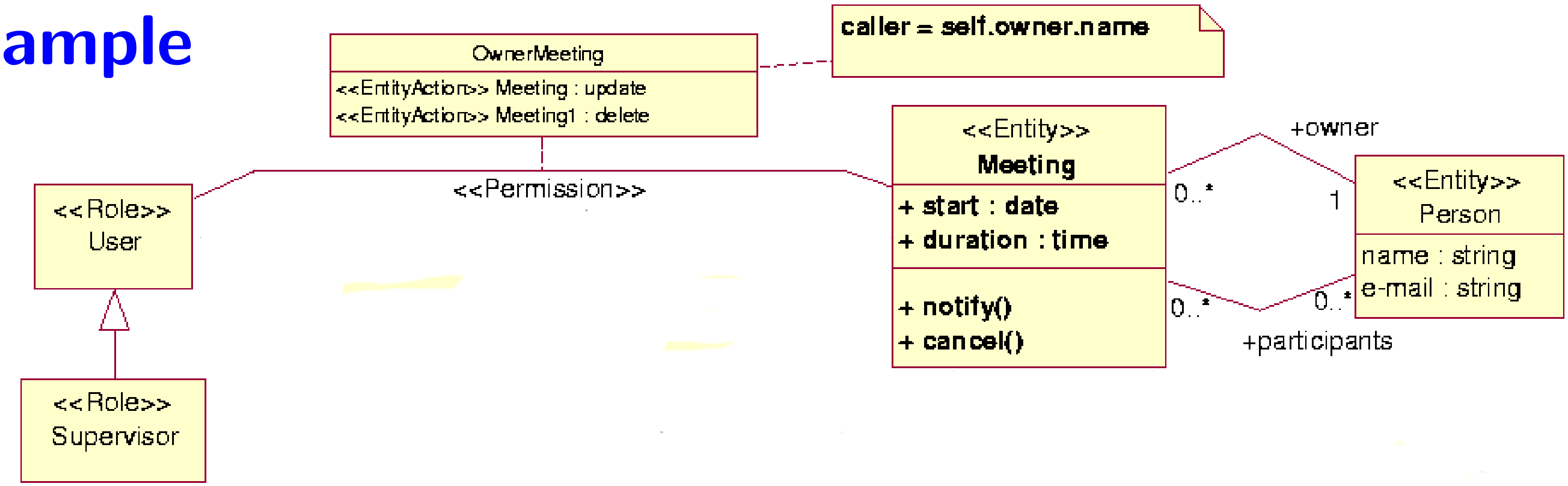
☞ **Generating .NET infrastructures.**

.NET versus EJB (from the AC perspective)

- Like with EJB, the protected resources are the component methods.
- .NET also supports both declarative and programmatic access control.
- Declarative access control is not configured in deployment descriptors, but by "attributes" of the methods, which name the allowed roles.
- Programmatic access control is conceptually very similar to EJB. For our purposes, the differences are only in the method names.

☞ Transformation function must be changed only slightly.

Example



generates the following C#-code:

```
[SecurityRole("User")]
[SecurityRole("SuperVisor")]
public void setStart(Date start){
if (!((ctxt.isCallerInRole("User")
          || ctxt.isCallerInRole("Supervisor"))
        && ctxt.OriginalCaller.AccountName ==
getOwner().getName()))
 throw new UnauthorizedAccessException("Access
denied.");
...
}
```

First two lines are “attributes”, naming the allowed roles.

Road Map

- Motivation and objectives
- Background
- Secure components
- Semantics
- Generating security infrastructures
- ☞ **Secure controllers**
- Experience and conclusions

What are Controllers?

- A controller defines how a system's behavior may evolve.
  Definition in terms of states and events, which cause state transitions.

- Examples
  - An application changes its state according to clicks on menu-entries in the user interface.
  - A washing machine goes through different washing/drying modes.
  - A control process that governs the launch sequence of a rocket.

- Mathematical abstraction: a transition system or some (hierarchical or parallel) variant.

Modeling Controllers

- Let's consider a language for modeling controllers for multi-tier architectures.
- A common pattern for such systems is the Model-View Controller.

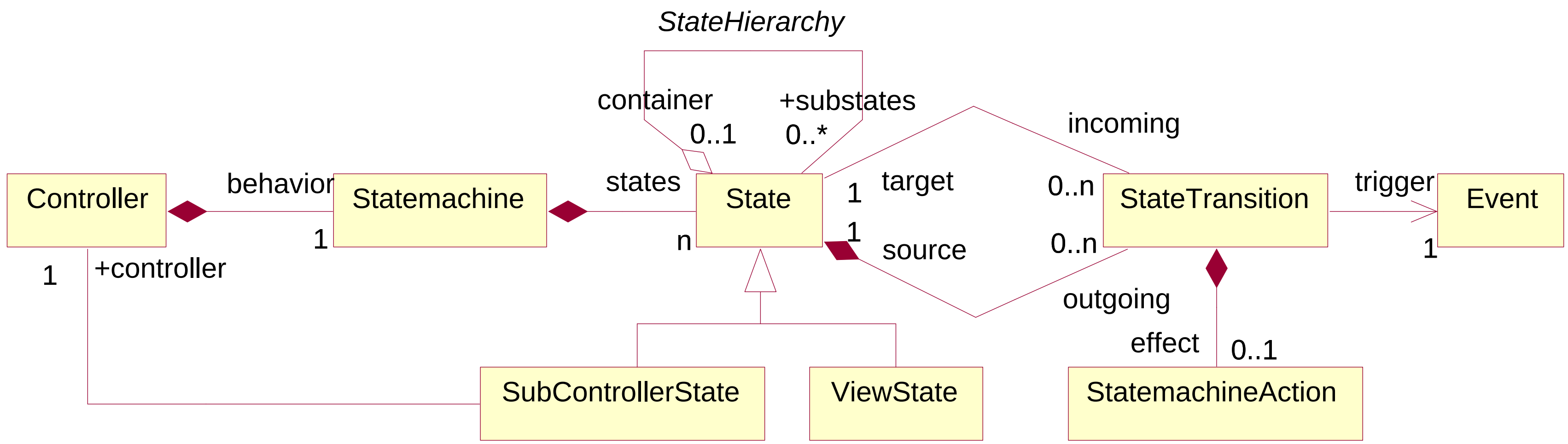
  Visualization tier: for viewing information. Typically within a web browser.

  Persistence tier: where data (model) is stored, e.g., backend data-base system.

  Controller tier: Manages control flow of application and dataflow between visualization and persistence tier.
- Our models must link "controller classes" with (possibly persistent) state with visualization elements.

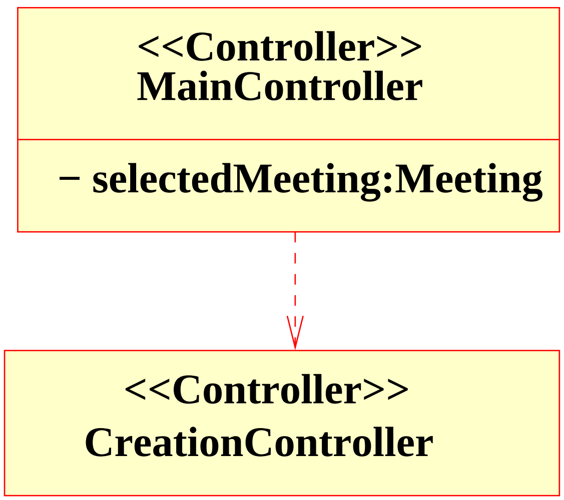
Abstract Syntax — ControllerUML

Metamodel (MOF):

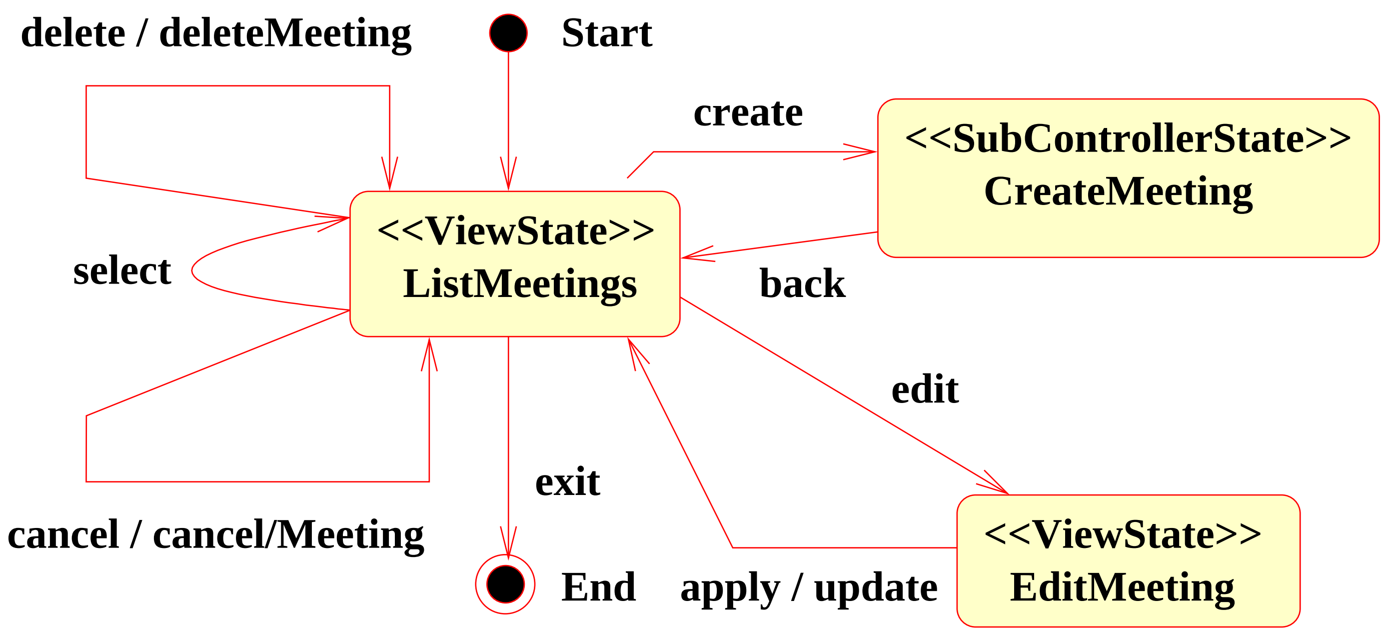


- A Statemachine formalizes the behavior of a Controller.
- The statemachine consist of states and transitions.
- Two state subtypes: SubControllerState refers to a sub-controller, ViewState represents an user interaction.
- A transition is triggered by an Event and the (optionally) assigned StatemachineAction is executed during the state transition.

Controller Example

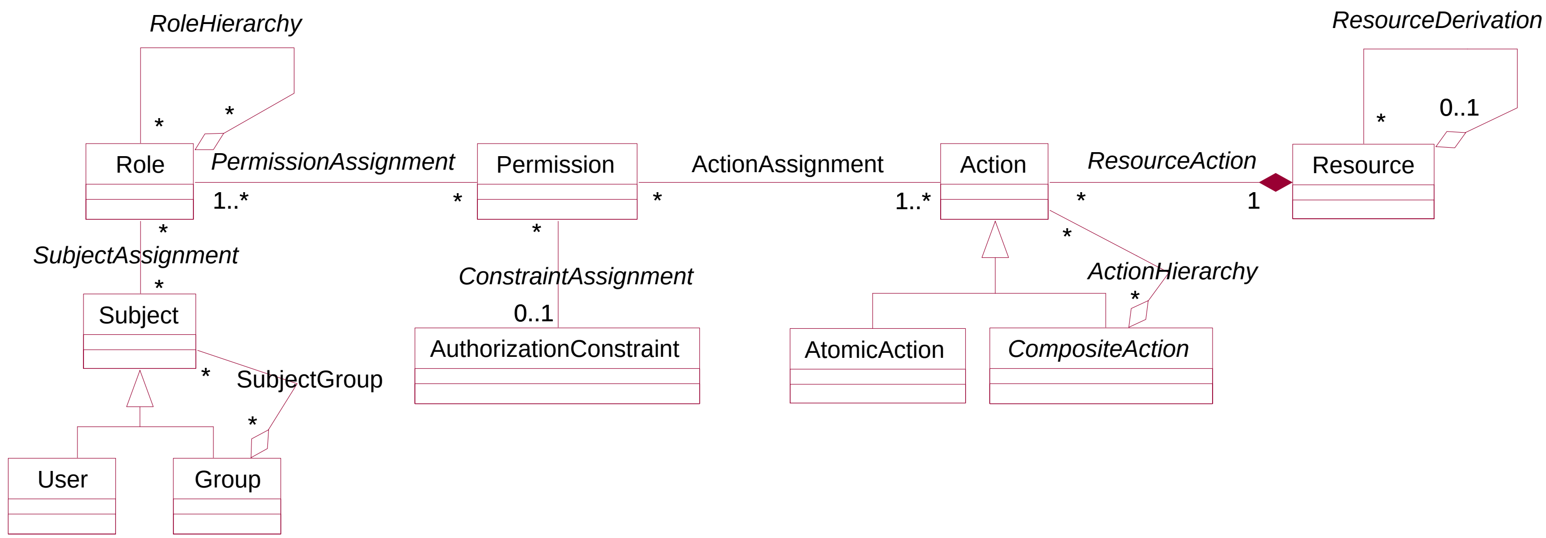


MainController's Statechart

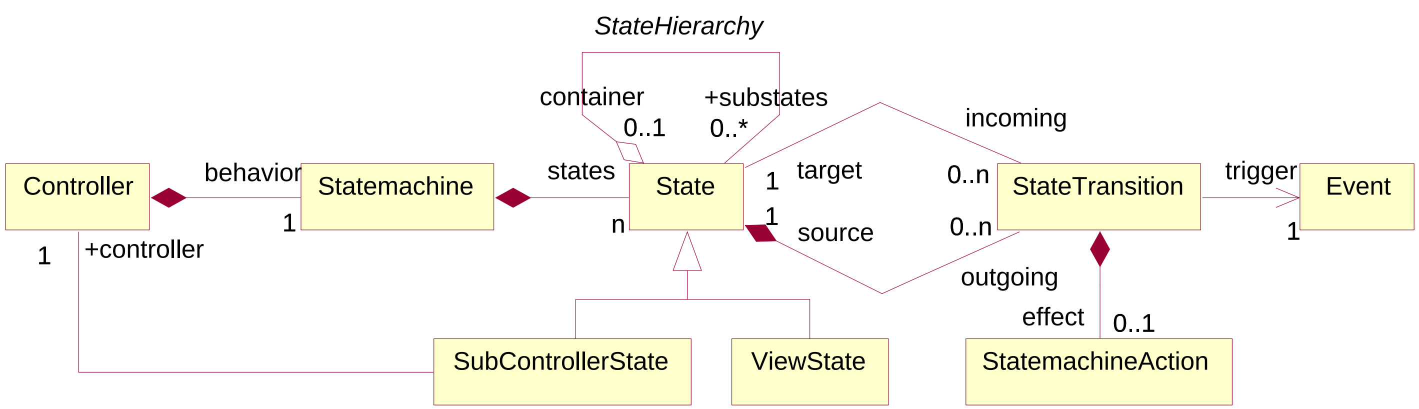


Dialect as a Bridge

- Security Modeling Language = SecureUML

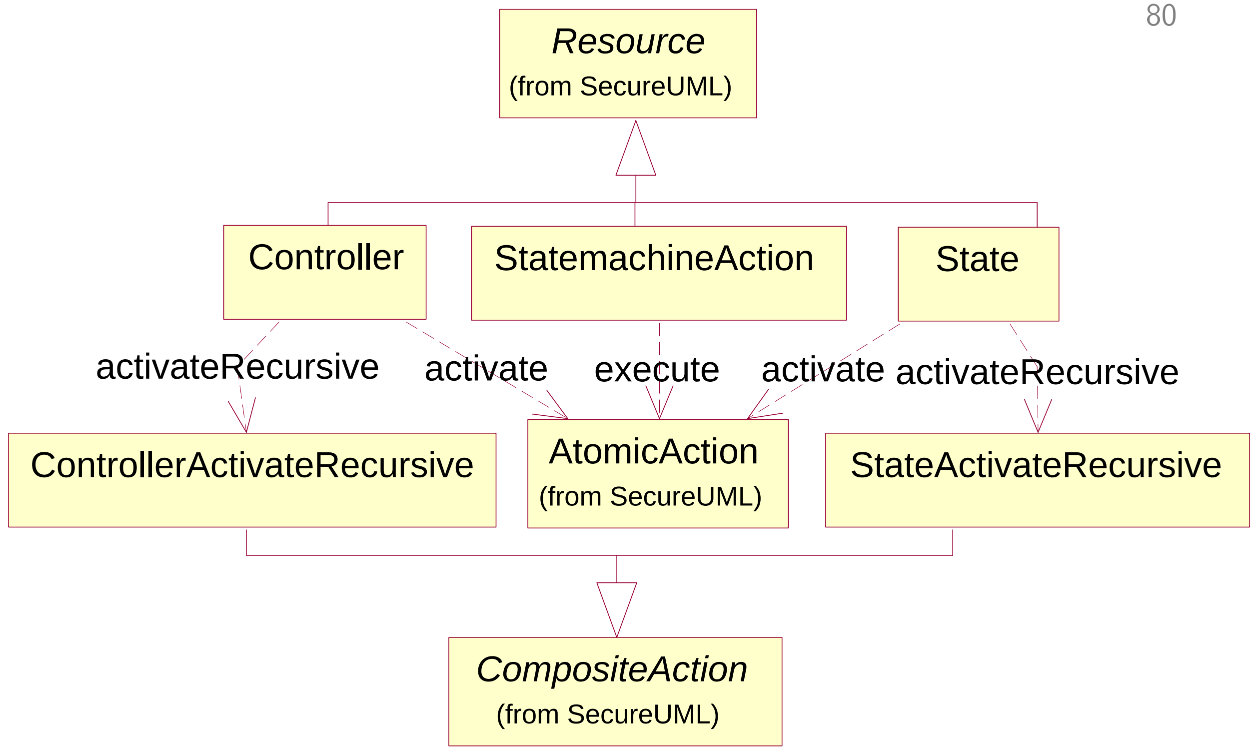


- System Design Modeling Language = ControllerUML



What are ControllerUML's protected resources? (States, Actions, ...?)

Dialect Definition



- Define resources and actions:

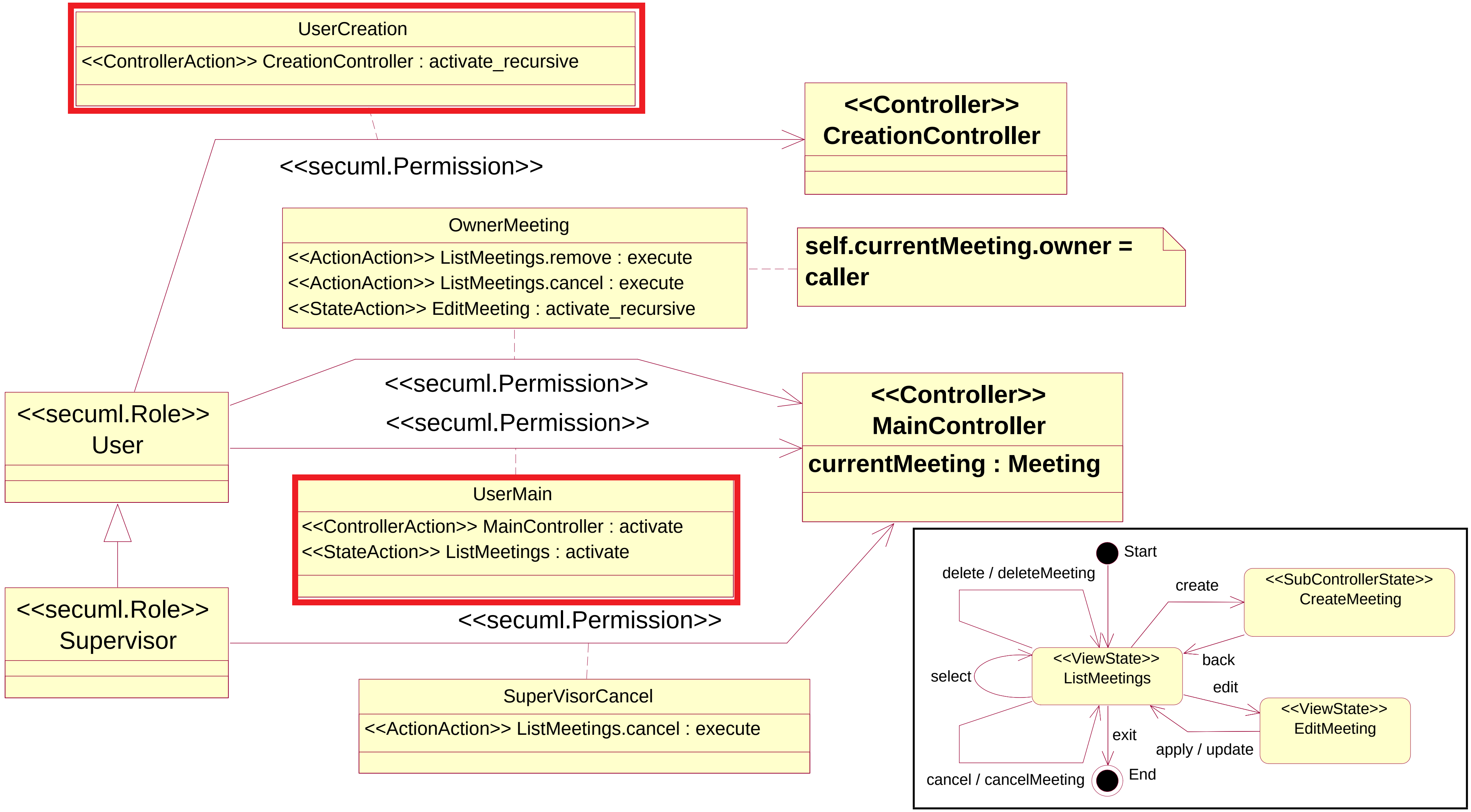
- Define action hierarchy:
  - State.activateRecursive: activate on the state, activateRecursive on all substates, and execute on all actions on outgoing transitions
  - Controller.activateRecursive: activate on the controller and activateRecursive on all states of the controller

Result is a vocabulary for defining permissions on both high-level and low-level actions.

Semantics

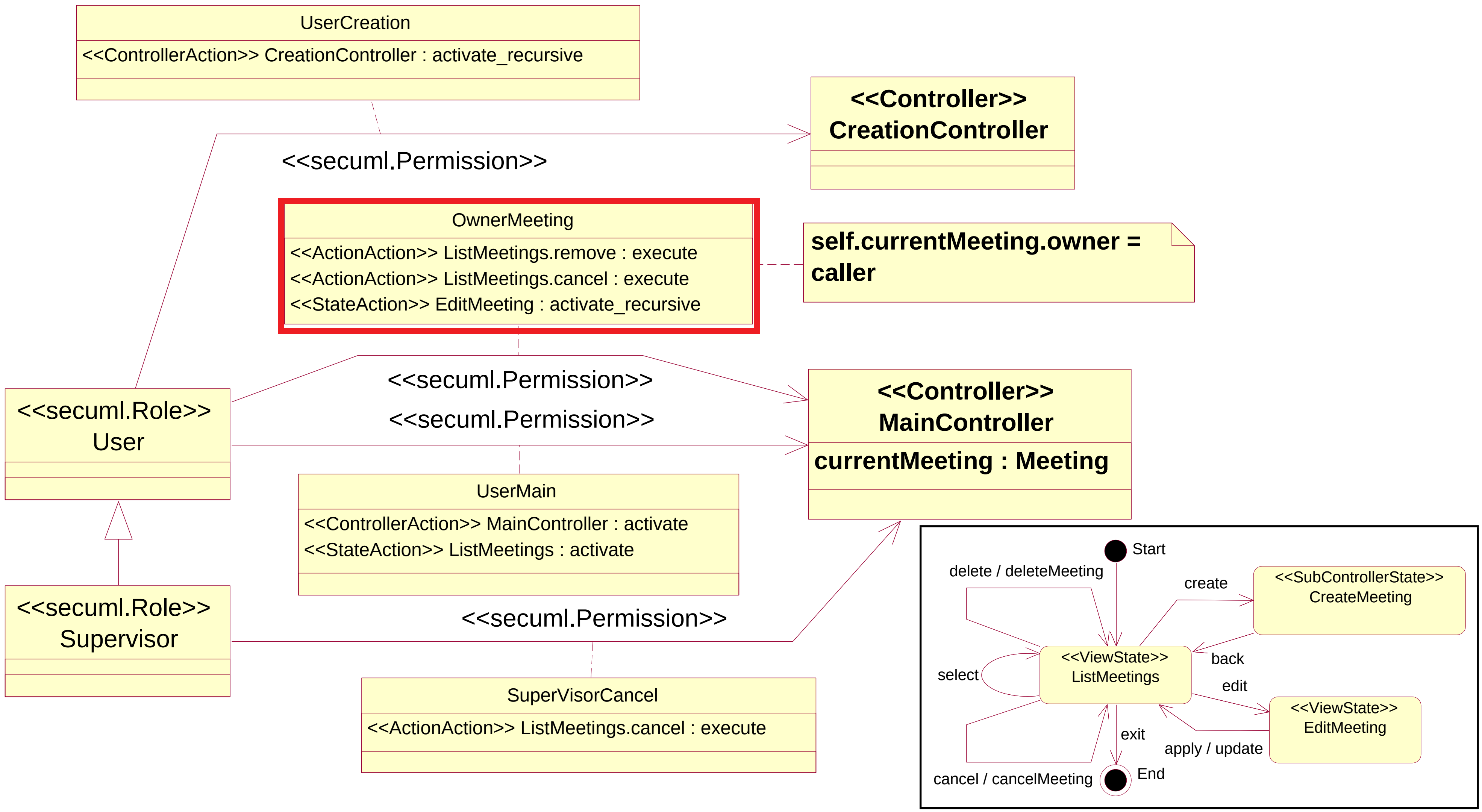
- It is not difficult to give a transition system semantics to a controller.
- Our general schema then provides a semantics for combination with SecureUML.
- See TOSEM paper for details.

Example Policy: Permissions



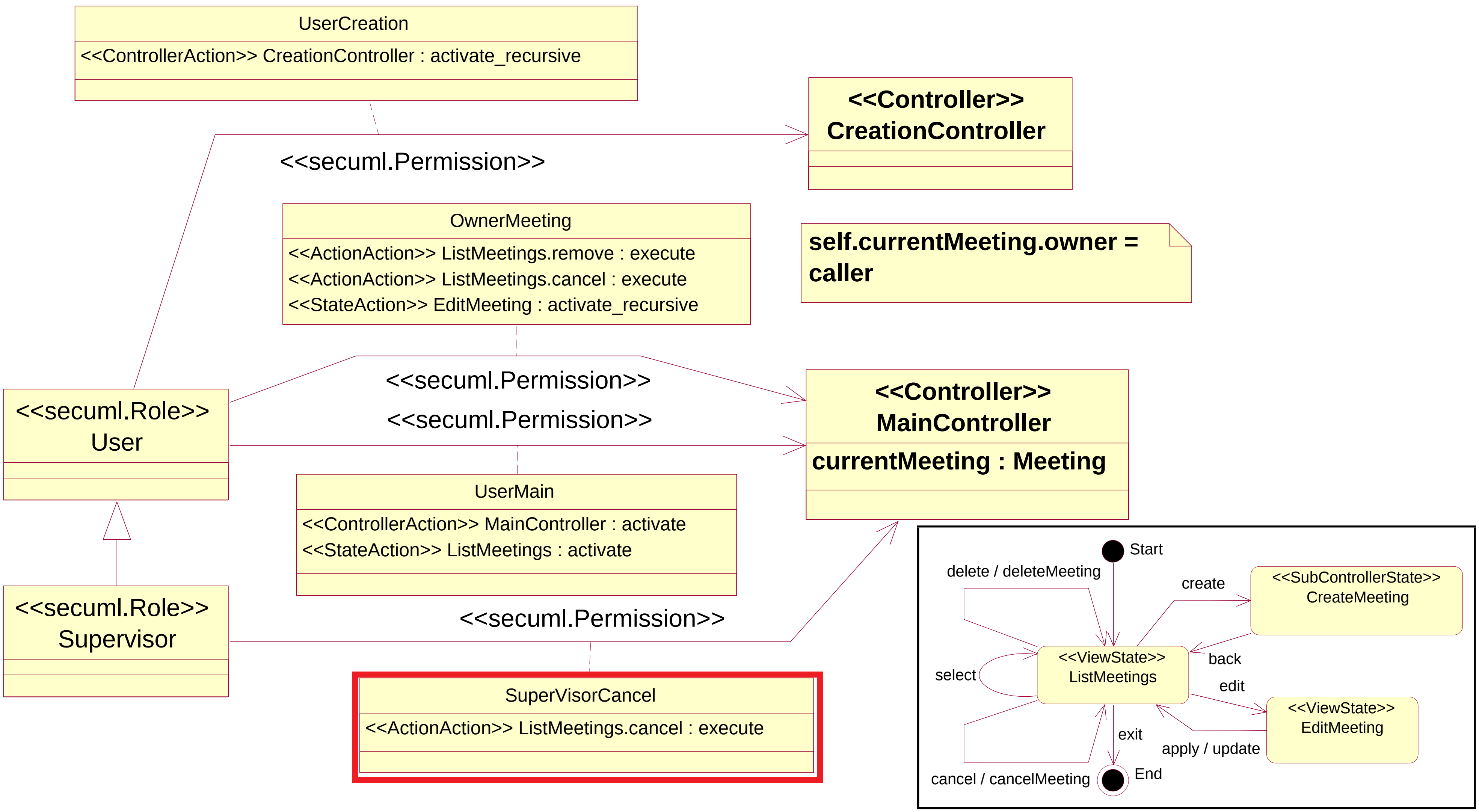
1. All users can create new meetings and read all meeting entries.

Example Policy: Permissions



2. Only the owner of a meeting may change meeting data and cancel or delete the meeting.

Example Policy: Permissions



3. However, a supervisor can cancel any meeting.

Generation (sketch)

- Generate web applications based on the Java Servlet platform.

  Each controller implemented as a servlet.

- Servlets process HTTP requests and create HTTP responses.
  - Support RBAC, but only for requests from outside web server.
  - Ill-suited for multi-tier (controller) based applications.
  - We overcome this using programmatic access control.

- Assertions added as preconditions to methods for process activation, state activation, and action execution.

- Tool generates complete controller and security infrastructure.

  Business logic and view element "stubs", for later elaboration.

Road Map

- Motivation and objectives
- Background
- Secure components
- Semantics
- Generating security infrastructures
- Secure controllers

☞ **Experience and conclusions**

Current Status

Foundational:

- Developed idea of Model Driven Security.
- General schema and various instances.

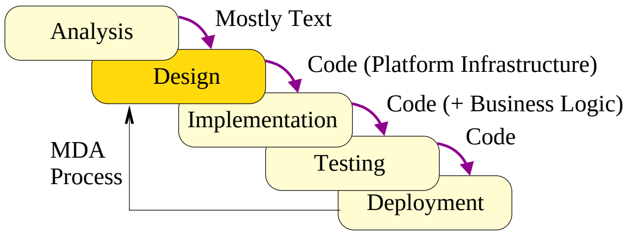
Practical/Tool: Prototype built on top of ArcStyler MDA Tool.

- Generators for J2EE (Bea EJB Server) and .NET.
- Industrial version developed by Interactive Objects Software GmbH.
- Also implemented in the SecureMOVA tool (IMDEA software).

Positive experience:

- In following, we briefly describe one of our case-studies: E-Pet Store.
- Standard J2EE example: an e-commerce application with web front-ends for shopping, administration, and order processing.

Pet Store Case Study



- Requirements analysis: Use Case Model identifying 6 roles (2 kinds of customers, 4 kinds of employees) and their tasks.

- Use Cases and their elaboration in Sequence Diagrams paved the way for the design phase.

  - 31 components
  - 7 front-end controllers
  - 6 security roles based on the Use Case roles.

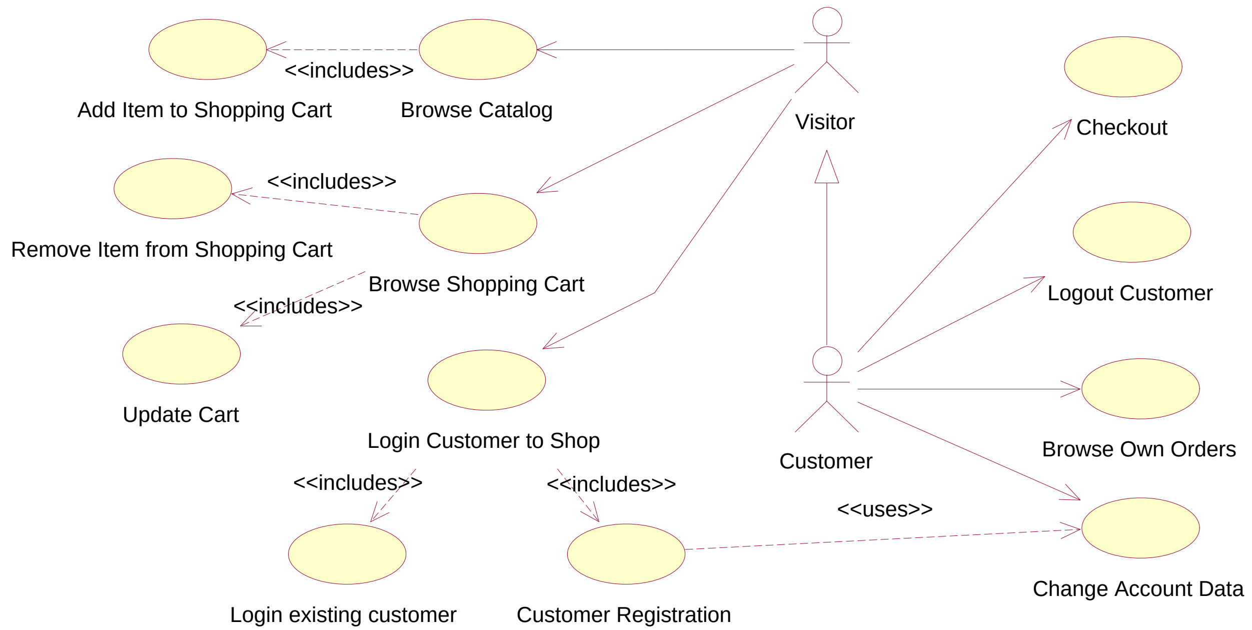
- Security policy based on principle of least privilege.

  Typical requirement: Customers need to read all catalog data, to update their own customer data, to create purchase orders, and to read their own purchase orders.

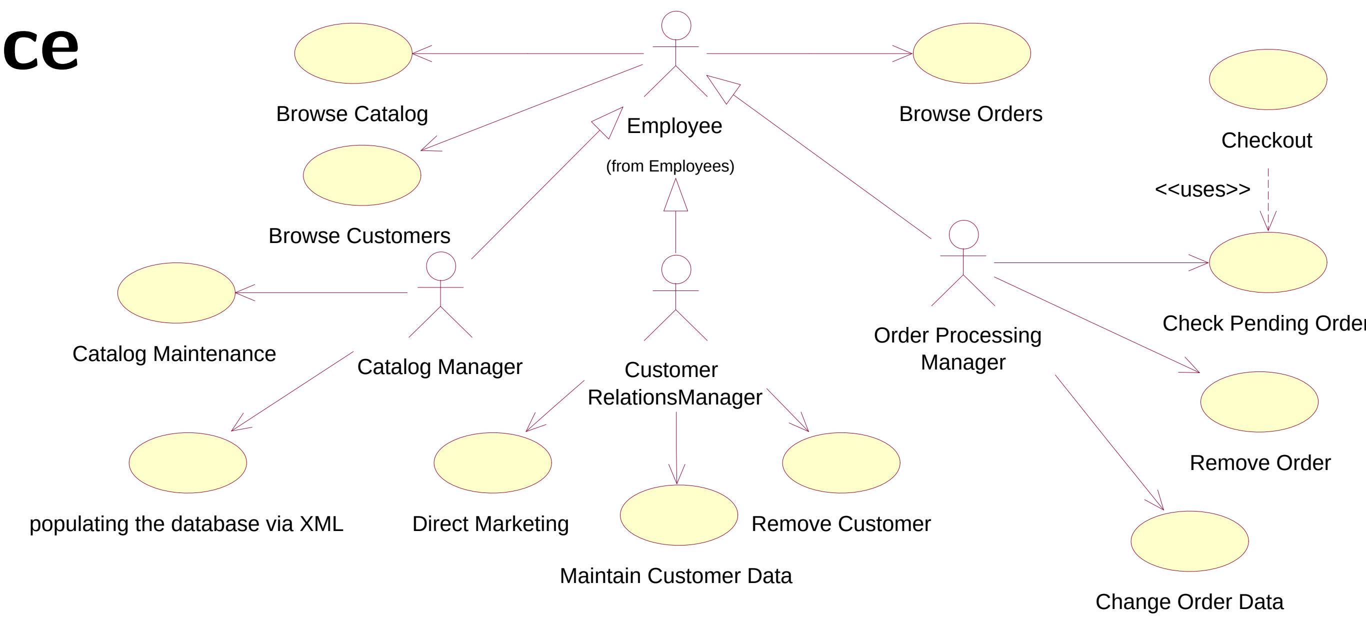
Let us look at a few snapshots from the model

Use Cases

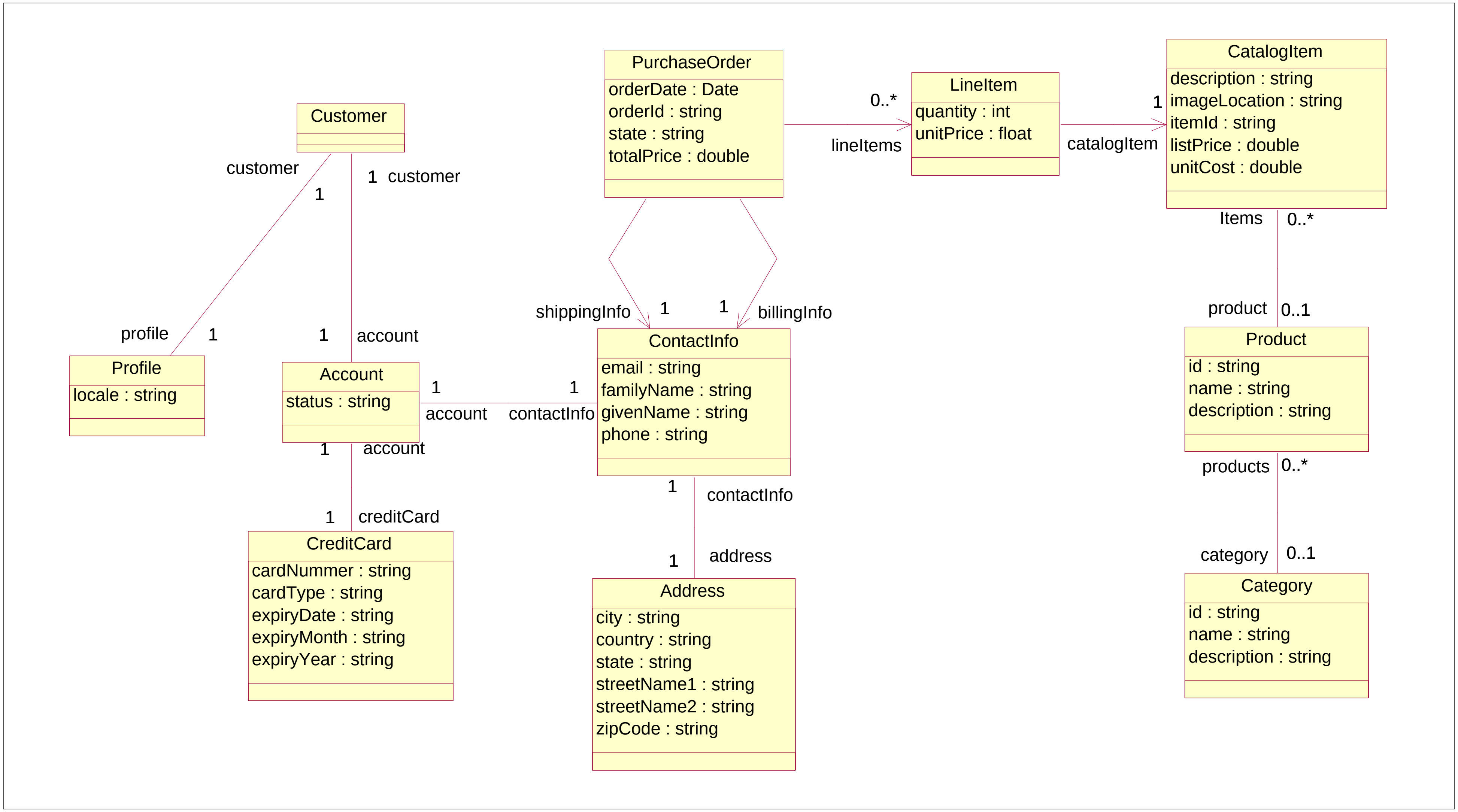
Shopping



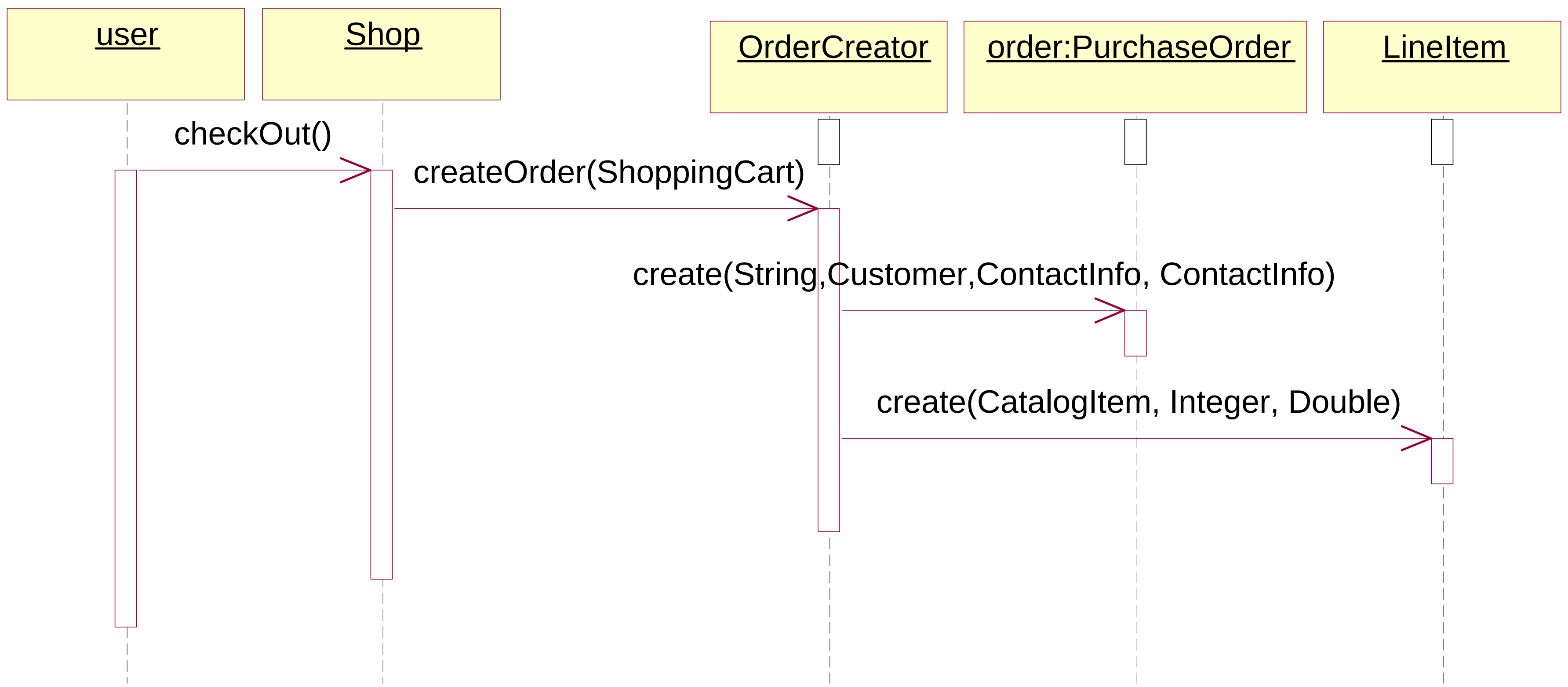
Back Office



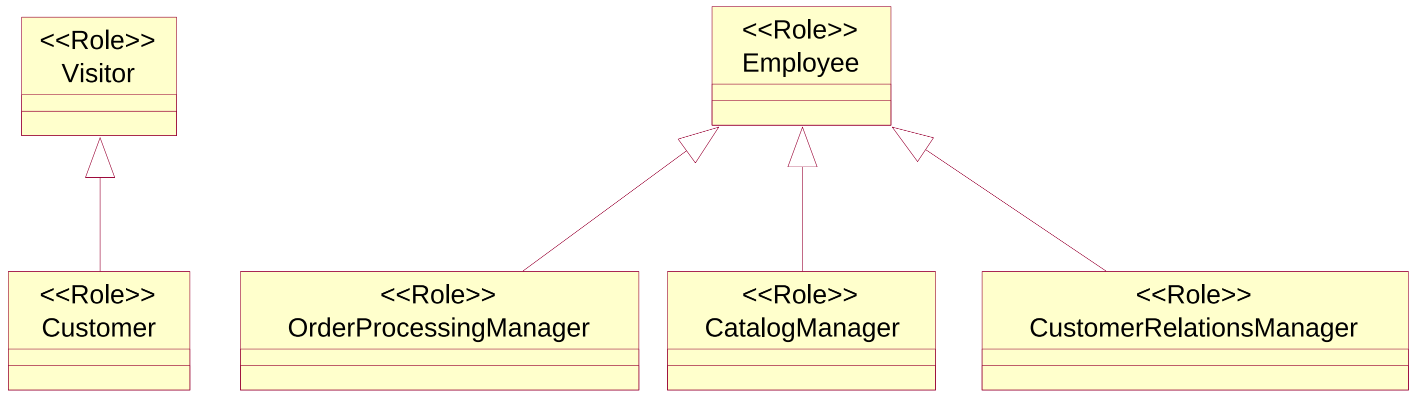
Component Model (partial)



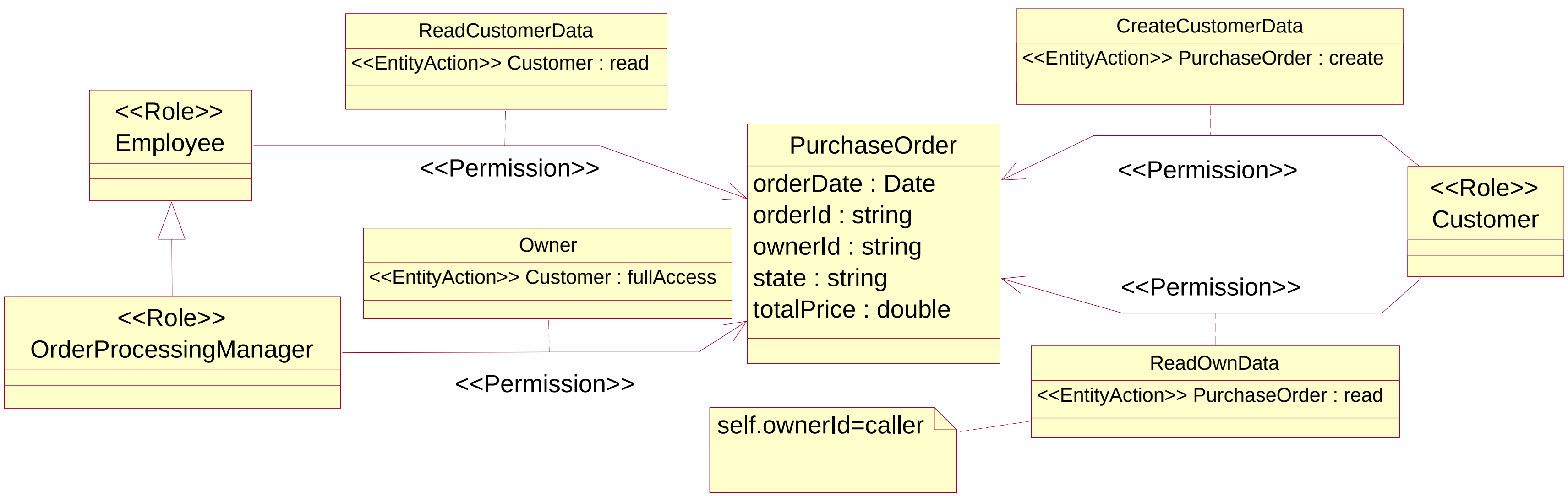
Sequence Diagram for Checkout Use Case



Role Model



Example of some permissions



Case Study — Evaluation

Model
6 roles
60 permissions
15 authorization constraints

→

System
5,000 lines XML (overall 13,000)
2,000 lines Java (overall 20,000)

Which would you rather maintain?

Evaluation (cont.)

- Expansion due to high-abstraction level over EJB.

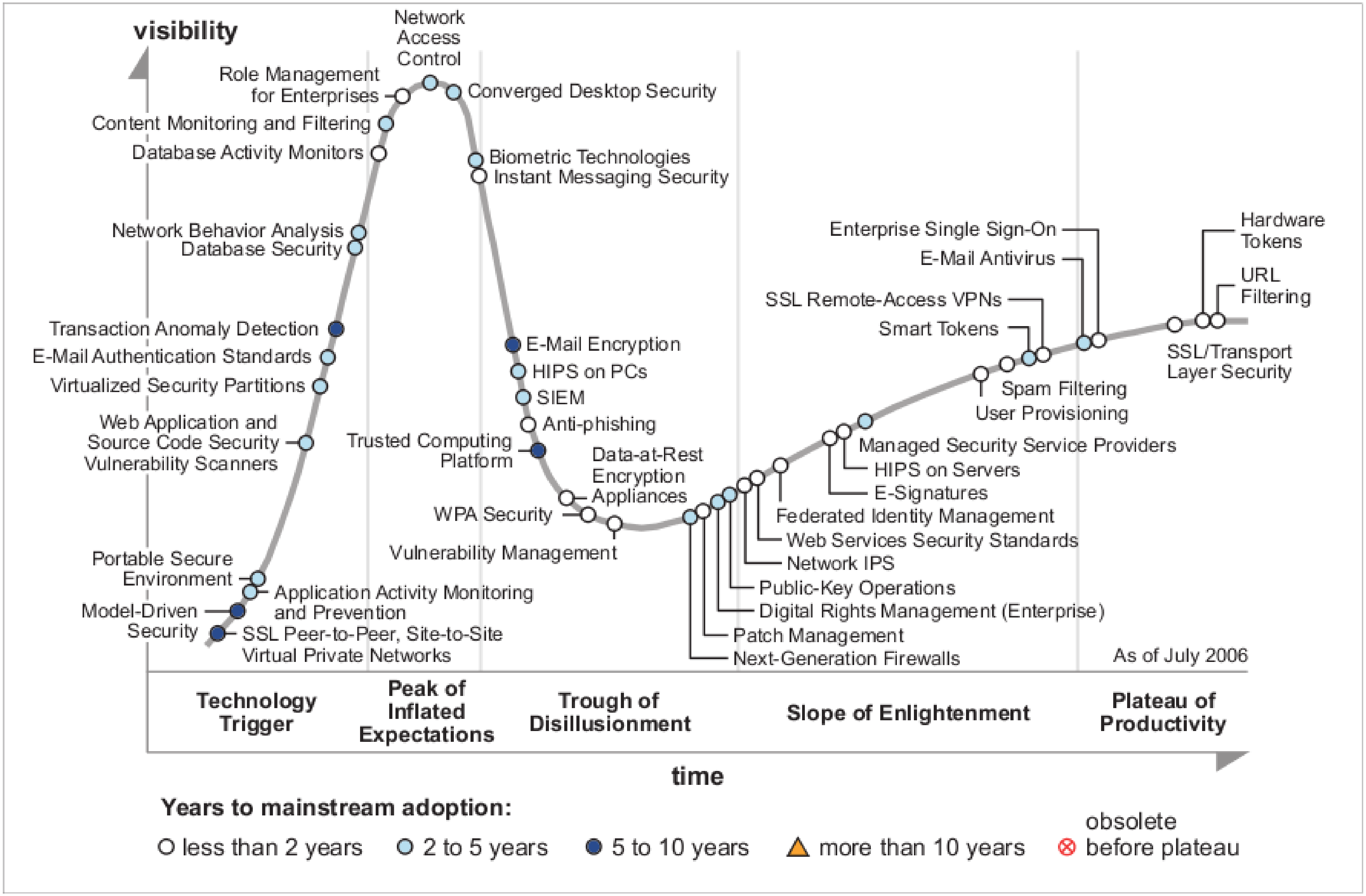
  Analogous to high-level language / assembler tradeoffs.
  Also with regards to comprehensibility, maintainability, ...

- **Claim:** Least privilege would be not be practically implementable without such an approach.

- Effort manageable: 2 days for designing access control architecture (overall development time: 3 weeks).

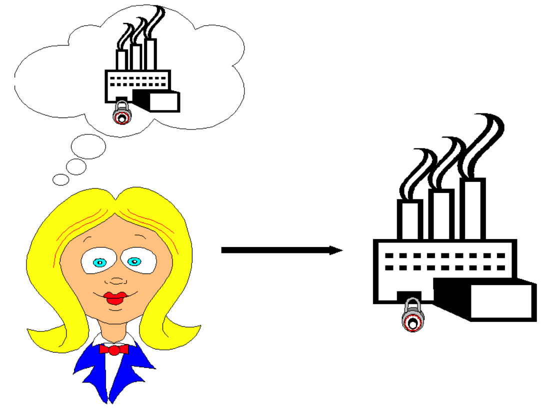
- MDS process provides conceptual support for building models
  - Fits well with a requirements/model-driven development process.
  - Provides a good transition from semi-formal to formal modeling.

Early Impact ...



Source: Gartner (July 2006)

Literature



- SecureUML: A UML-Based Modeling Language
  for Model Driven Security.
  Lodderstedt/DB/Doser, *UML 2002*.

- Model Driven Security for Process-Oriented Systems.
  DB/Doser/Lodderstedt, *SACMAT 2003*.

- Model Driven Security: From UML Models to Access Control Infrastructures.
  DB/Doser/Lodderstedt.
  ACM Transactions on Software Engineering Methodology January 2006.

- Automated Analysis of Security Design Models.
  DB/Clavel/Doser/Egea. *Information and Software Technology*, 2009.

- Automatic Generation of Smart, Security-Aware GUI Models.
  DB/Clavel/Egea/Schläpfer.
  International Symposium on Engineering Secure Software and Systems, 2010.

- A Decade of Model-driven Security
  DB/Clave/Egea
  16th ACM Symposium on Access Control Models and Technologies (SACMAT), 2011.