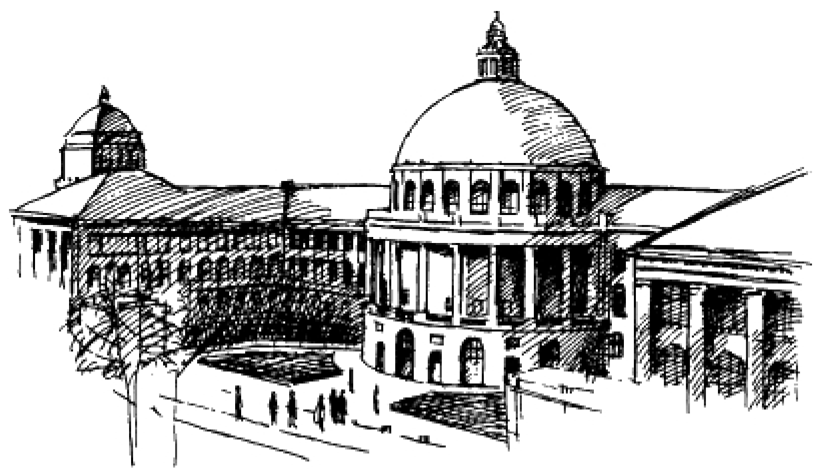
Model Driven Security

David Basin
ETH Zürich



Joint work with Jürgen Doser and Torsten Lodderstedt

Talk Objectives

Present a methodology for automatically constructing 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 experiments positive.

Talk Objectives

Present a methodology for automatically constructing 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.

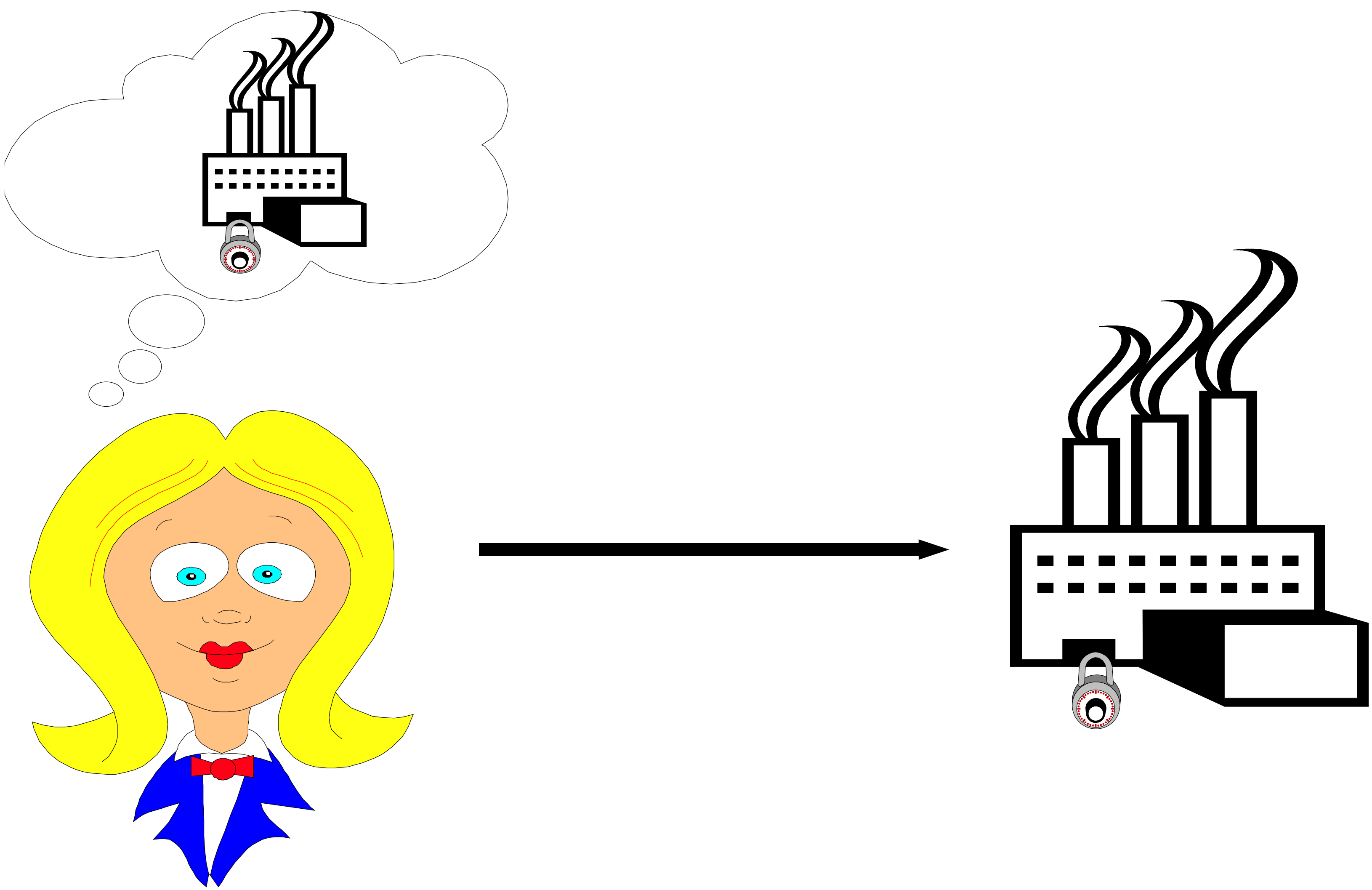
Scales: Initial experiments positive.

Submessage: formal and semiformal can live harmoniously together and the results can be practically useful.

Road Map

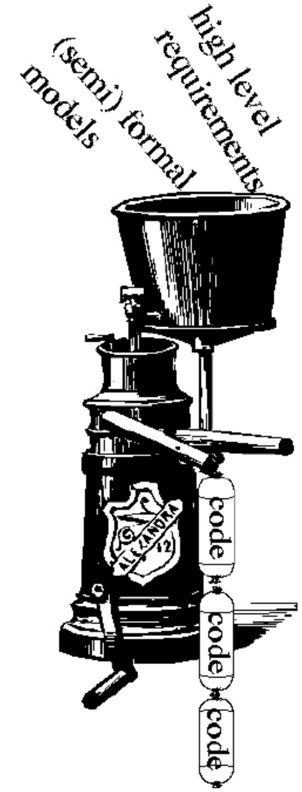
Mon	Motivation and objectives
Mon	Background
Tues	Secure components
Tues	Semantics
Thurs	Generating security infrastructures
Thurs	Secure controllers
Fri	Experience, demonstration, and conclusions

Motivation



How do we go from requirements to secure systems?

From Requirements to Systems

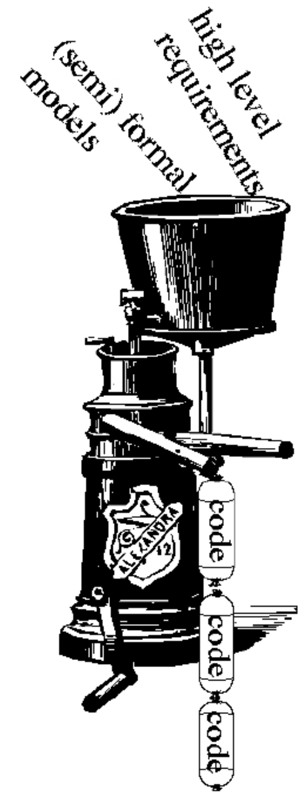


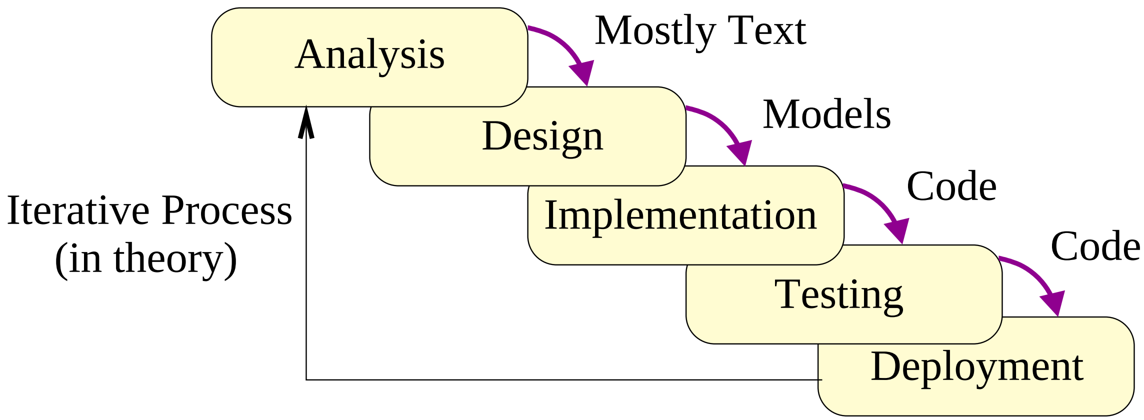
- Ideally: Automated synthesis from specifications.
  - The Holy Grail of Software Engineering!
  - But problem is not recursively solvable.

From Requirements to Systems

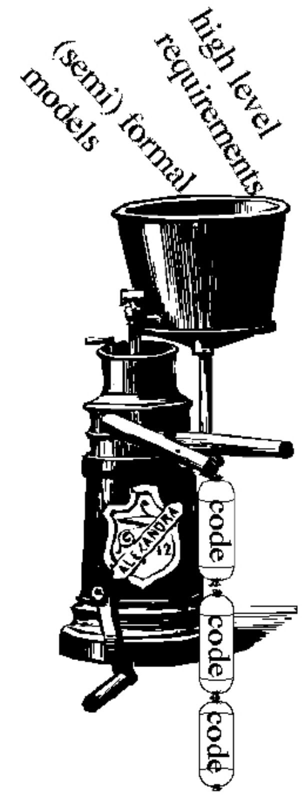
- Ideally: Automated synthesis from specifications.
  - The Holy Grail of Software Engineering!
  - But problem is not recursively solvable.

- As described by process models.



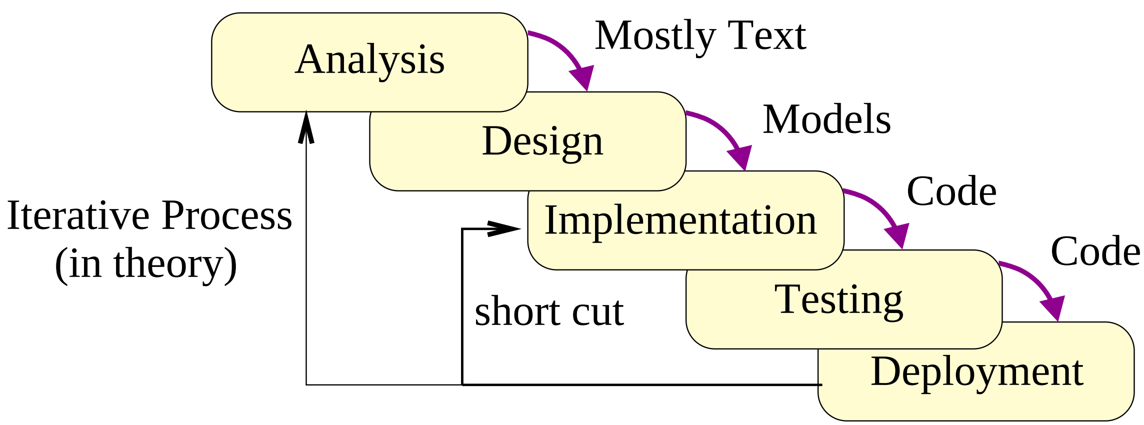


From Requirements to Systems



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

- As described by process models.

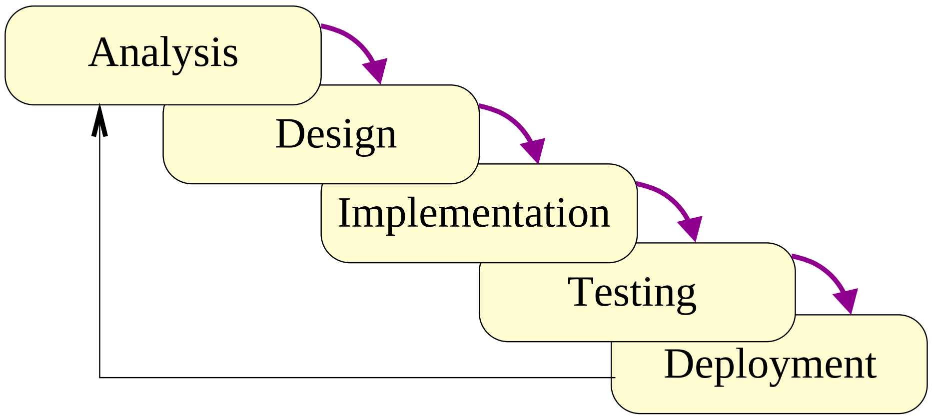


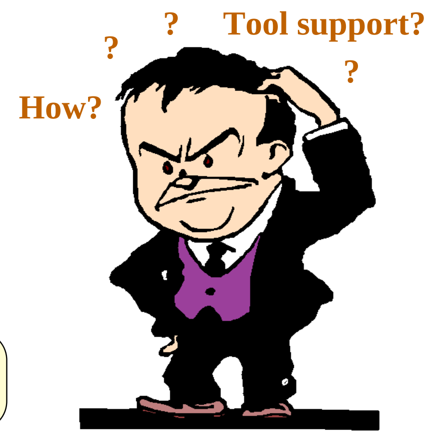
- In practice: code-and-fix.

  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

An 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:

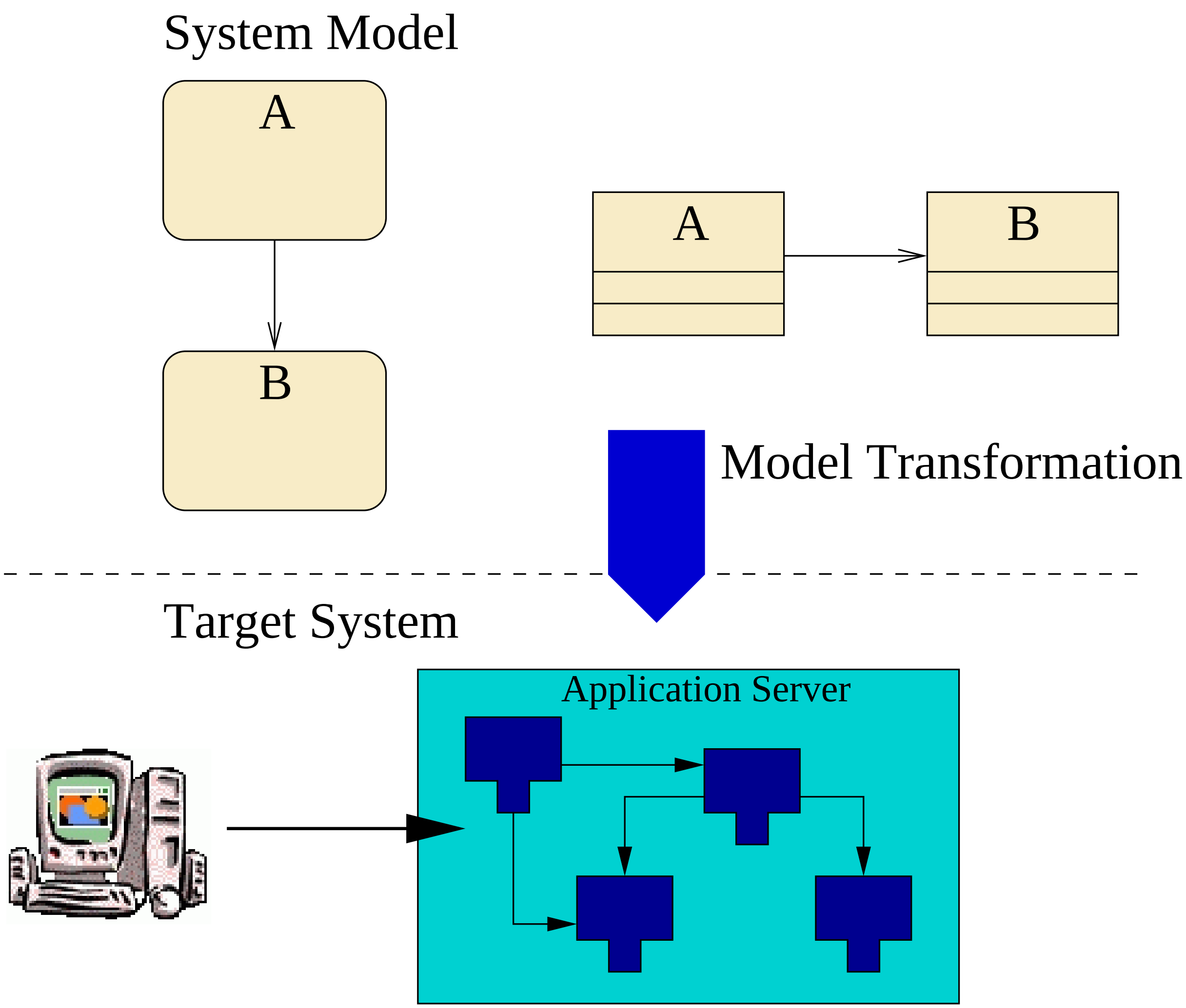
1. All users can create new meetings and read all meeting entries.
2. Only owners may change meeting data, cancel meetings, or delete meeting entries.
3. However, a supervisor can cancel any meeting.

Example — Some Questions

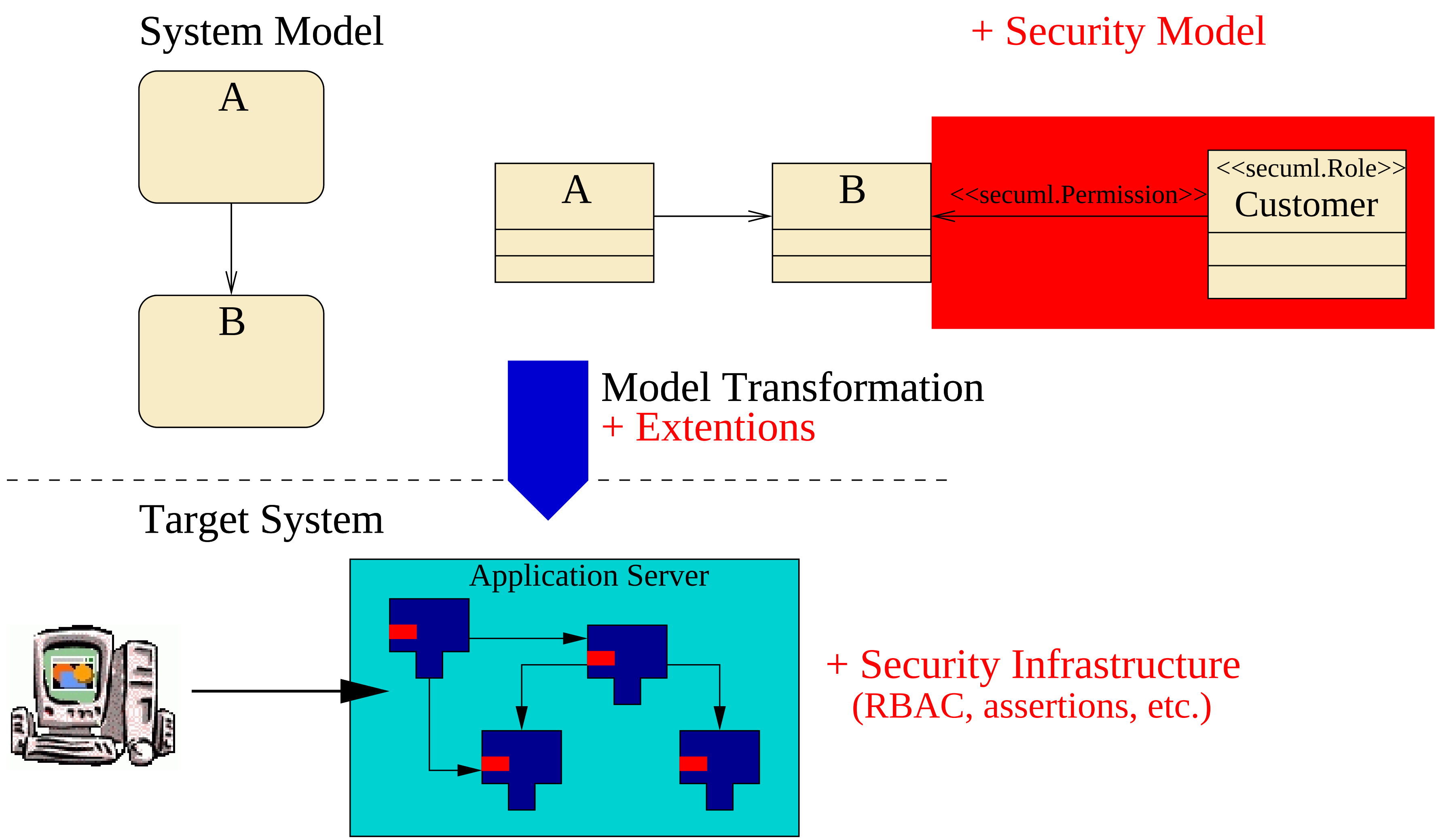
- How do we formalize both kinds of requirements?
- How are requirements refined into multi-tier architectures with support for GUIs, controllers, database back ends ...
- Can this be done in a way that supports modern standards/technology for modeling (UML), middleware (EJB, CORBA, ...), and security?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

The methodology and the tool presented in this course will address all of these concerns.

Approach: Specialize Model Driven Architecture

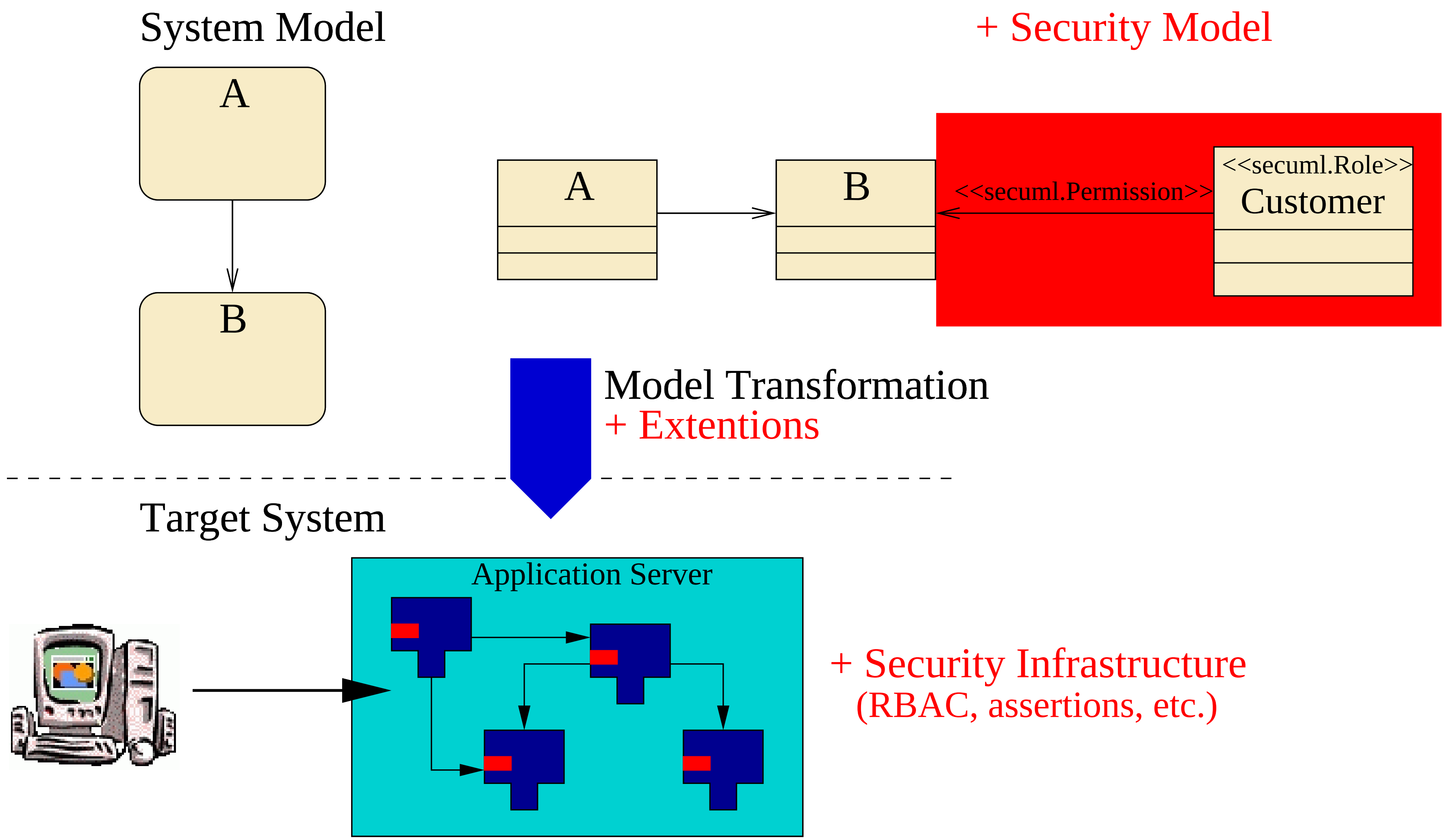


Approach: Specialize Model Driven Architecture



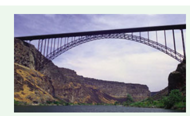
to Model Driven Security.

Approach: Specialize Model Driven Architecture



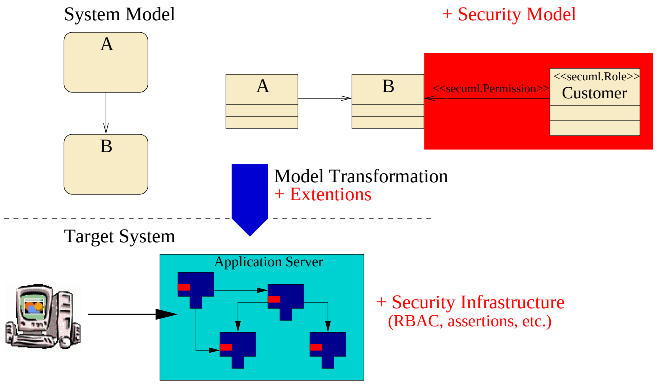
to Model Driven Security.

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 (e.g., 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, demonstration, and conclusions

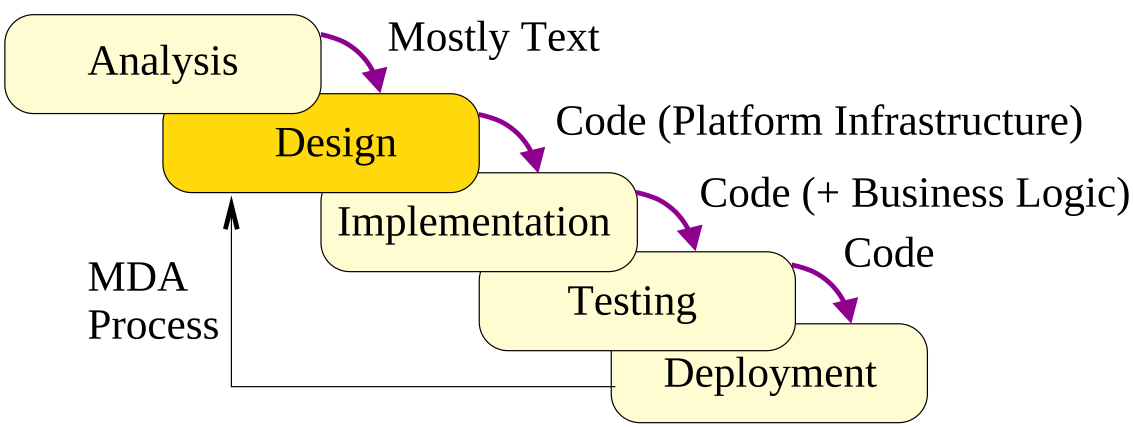
Background

☞ **Model Driven Architecture**

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

MDA: the Role of Models

- A model presents a view of the system 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 an emerging Object Management Group standard.
  - Standards are political, not scientific, constructs.
  - They are valuable, however, 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

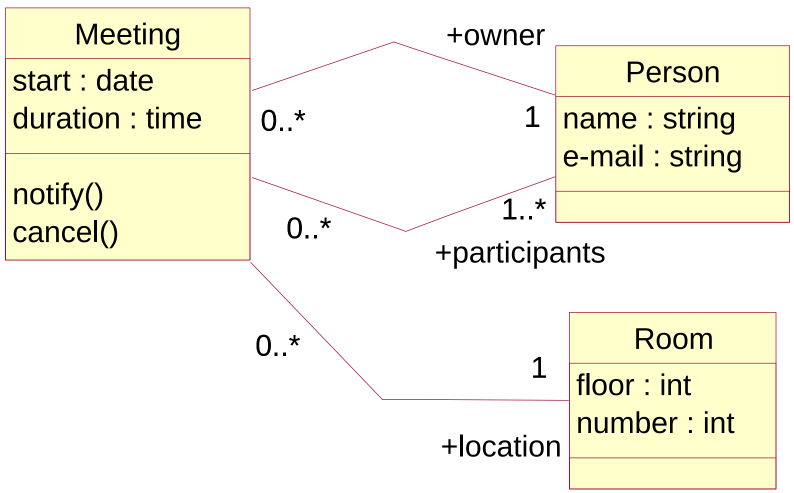
UML

- Family of 9 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

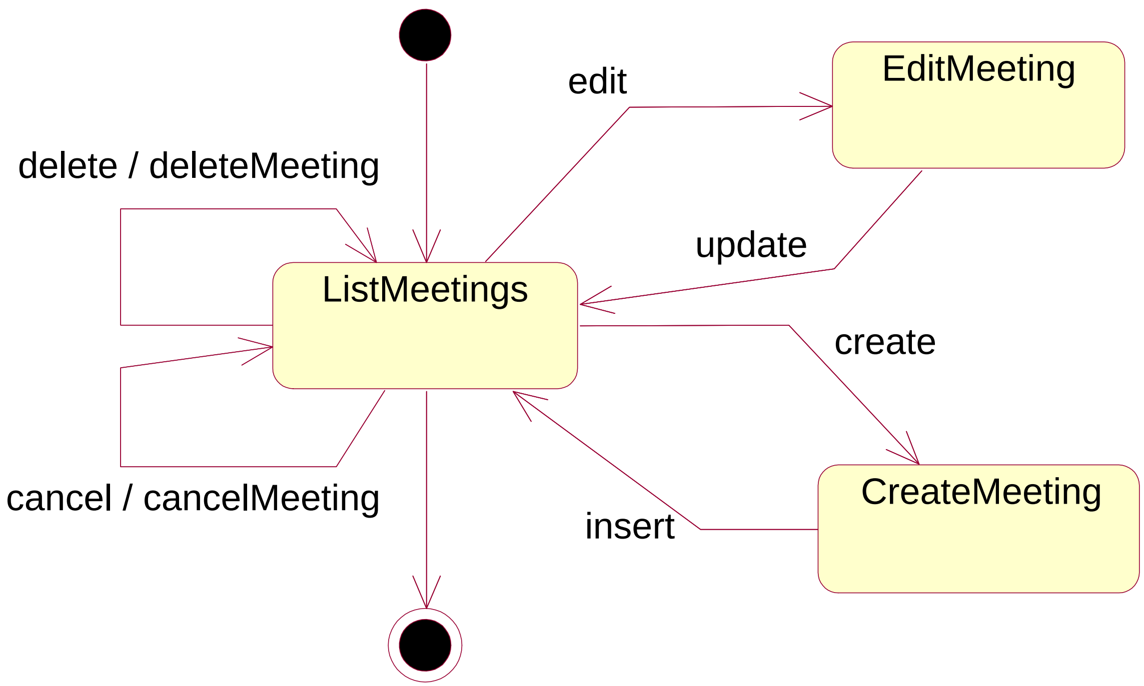
Describe structural aspects of systems. A class formalizes a set of objects with common services, properties, and behaviors. Services are described by methods and properties by attributes and associations.



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, however, 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

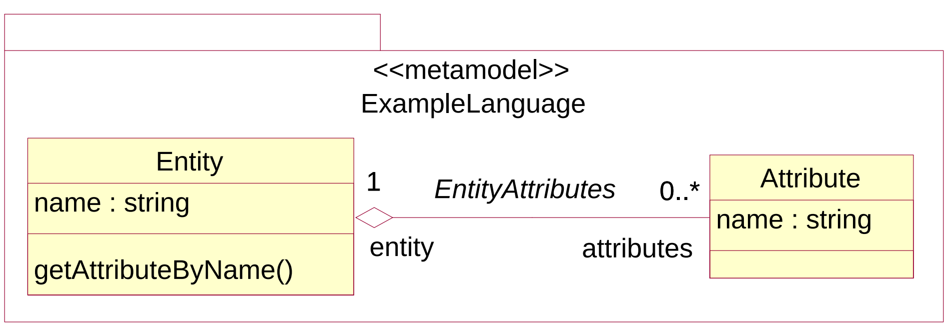
- A Metamodel defines the (abstract) syntax of other models.

  Its elements, metaobjects, describe types of model objects.

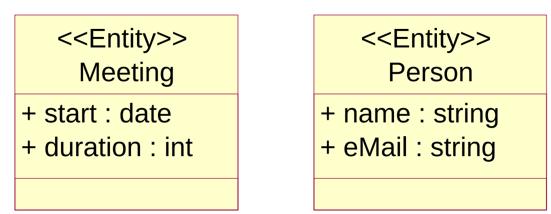
- MOF is a standard for defining metamodels.

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

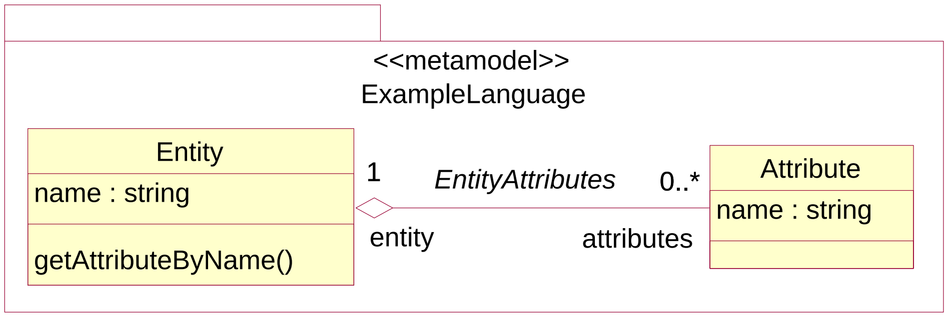


M1

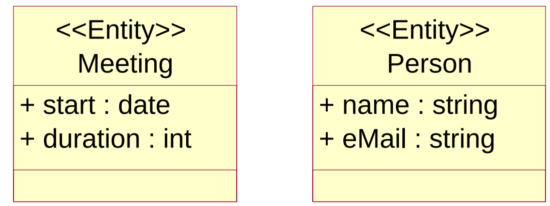


2) Metamodeling (cont.)

M2



M1



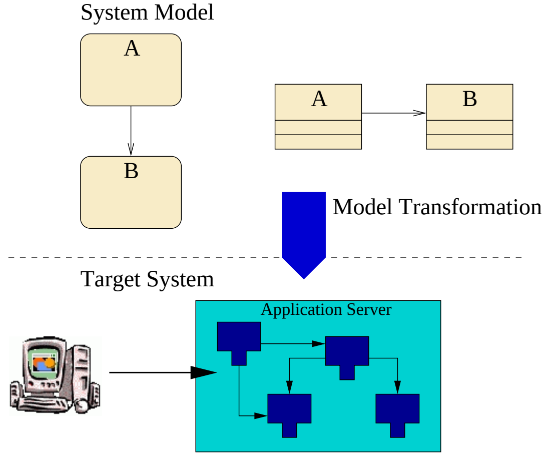
- Abstract syntax of metamodels defined using MOF.
  - Metamodels may be defined using UML notation.
  - Supports OO-metamodels, using concepts like subtyping.
- Concrete syntax of DSL defined by a UML profile.
- MOF/UML tools automatically translate models in concrete syntax into models in abstract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

Background

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

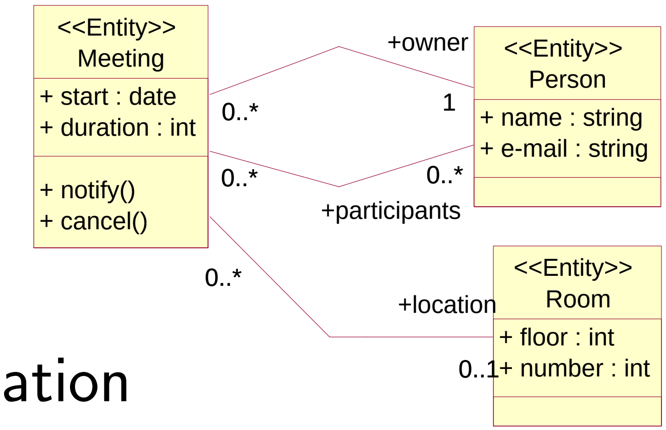
☞ **Code generation**

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 ↦ EJB component with implementation
  class, interfaces (local, remote, home, ...),
  factory method *create*, finder method *findByPrimaryKey*, ...

- Entity Attribute ↦ getter/setter methods

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

- Entity Method ↦ method stub

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

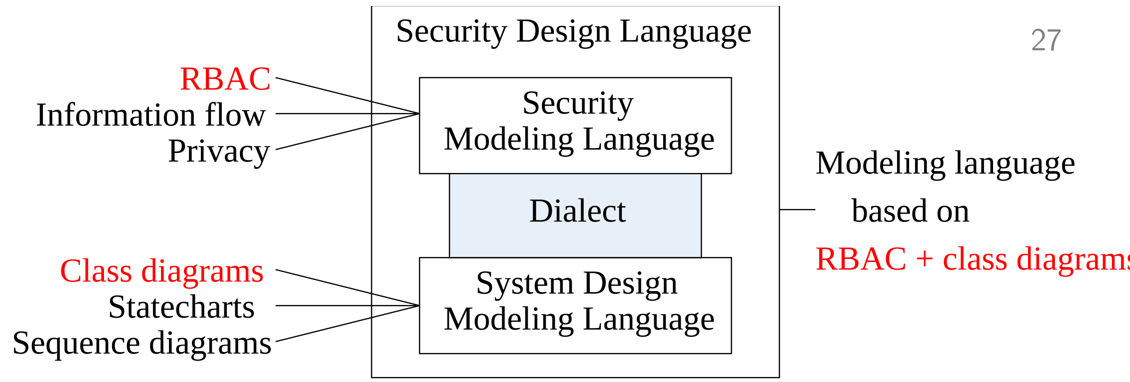
- Association Ends ↦ schema for maintaining references

```
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, demonstration, and conclusions

Context: Models and Languages



- A Security Design Language glues two languages together.

  Approach open (modulo some minimal 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

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

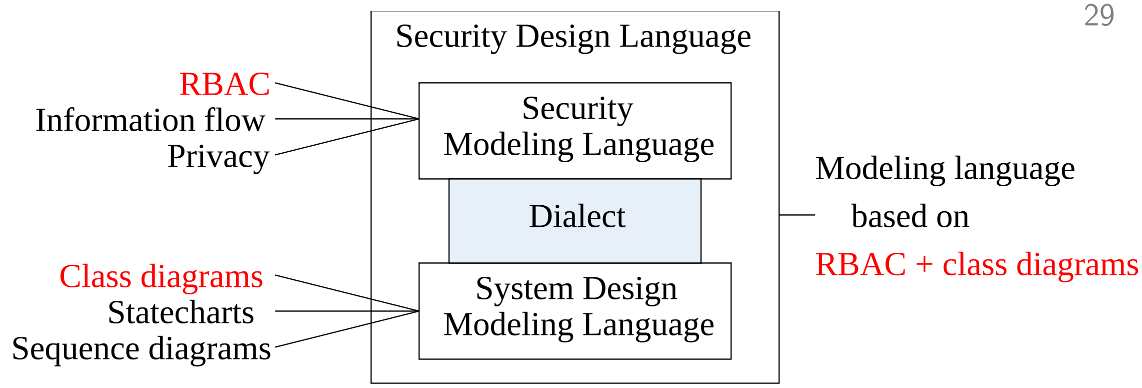
Secure Components

☞ 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 concern 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.

- These can be formalized as Access Control Policies detailing which subjects have rights (privileges) to read/write which objects.

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

- We will focus on Access Control Policies/Mechanisms in following.

Access Control

- Two variants 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.

- Role Based Access Control is a commonly used declarative model.
  - Roles are used to group privileges.
  - Other additions (e.g., hierarchies) are possible.

- These are often combined to make stateful decisions, 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: access control amounts to 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

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

i.e.,

$$\text{AC} := \{(u,p) \in \text{Users} \times \text{Permissions} \mid \exists r \in \text{Roles} : (u,r) \in \text{UA} \wedge (r,p) \in \text{PA}\} .$$

- Example:

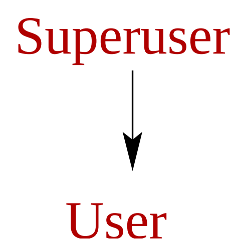
User	Role
Alice	User
Alice	Superuser
Bob	User
John	User

Role
User
Superuser

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

Role-Based Access Control

- Benefits of RBAC:
  - Roles model a basic abstraction (job/function)
  - Reduces complexity of access control policies (scalability)
- RBAC-Extensions:
  - Role hierarchy (for $\geq$ a partial order):



$$AC := PA \circ \geq \circ UA$$

Intuitively: larger roles inherit permissions from all smaller roles

  - Hierarchies on users (UA) and permissions (PA).
  - Authorization Constraints: predicates used to make stateful access control decisions, 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 EUR."

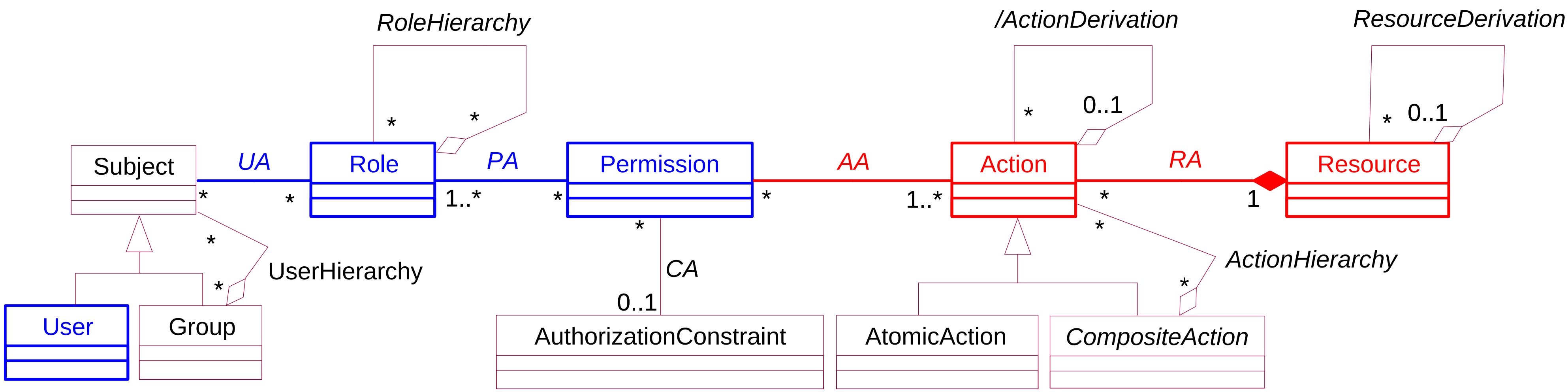
Secure Components

- Role-Based Access Control
- ☞ **Generalization to SecureUML**
- Component modeling and combination

SecureUML – Syntax

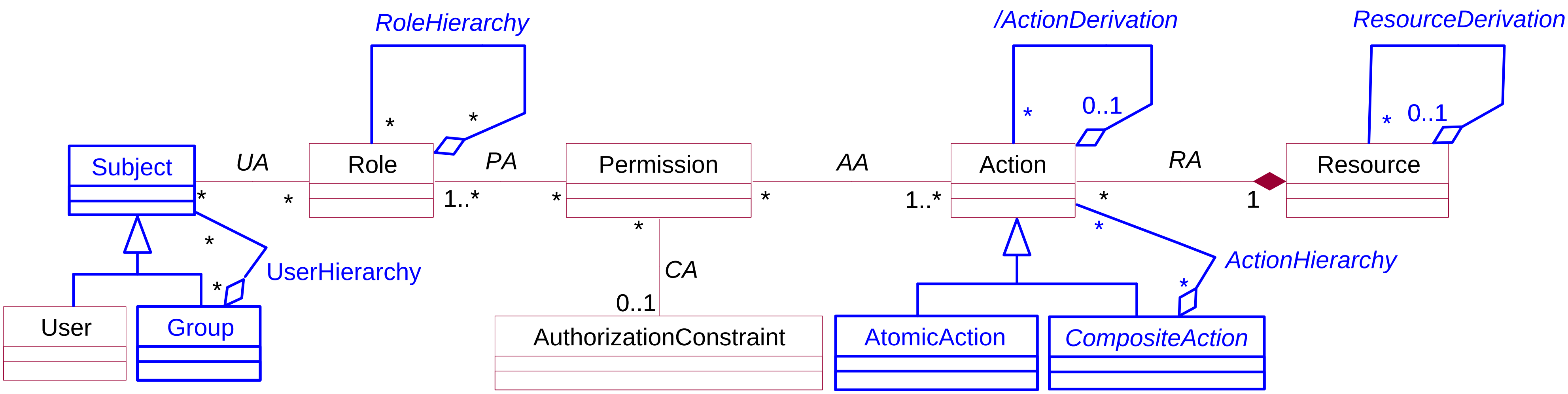
- Abstract syntax defined by a MOF metamodel.
- Concrete syntax based on UML and defined with a UML profile.
- Syntax and semantics based on an extension of RBAC.
- The key idea:
  - Access Control formalizes the permissions to perform actions on (protected) resources.
  - We leave these open: types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

Users, Roles and Typed Permissions



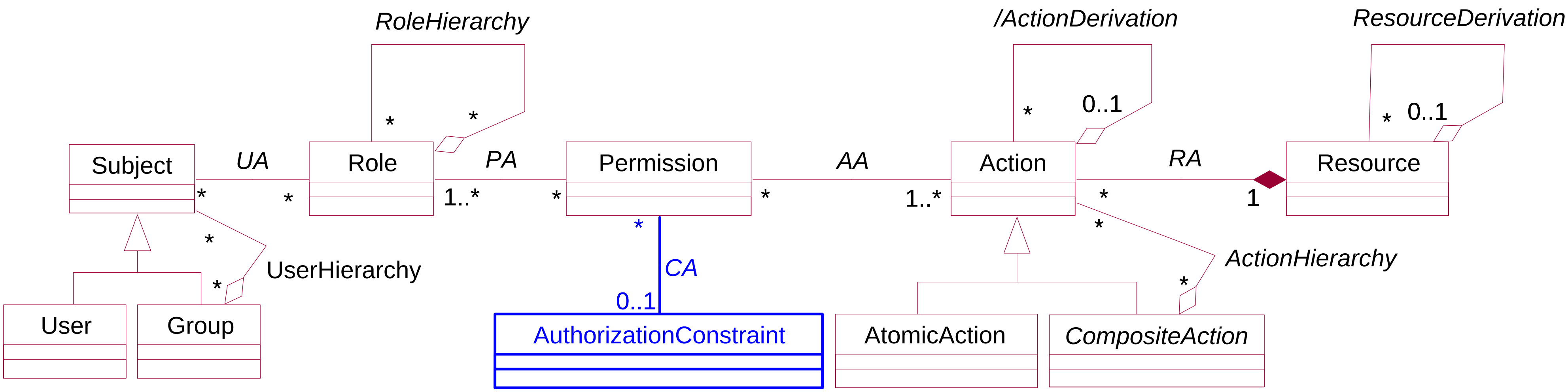
- Left hand part: essentially Standard RBAC
- Right hand part: permissions are factored into the ability to carry out actions on resources.
  - Resource is the base class of all model elements representing protected resources (e.g. "Class").
  - Actions of a "Class" could be "Create", "Read", "Delete"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

Hierarchies over Users, Roles and Actions



- UserHierarchy: Users (and groups) are organized in groups.
- RoleHierarchy: Roles can be in an inheritance hierarchy.
- ActionHierarchy (Sub/Super), e.g. "FullAccess" is a super-action of "Read" $\mapsto$ Intuitive semantics for high-level actions and reduced development effort for generators (rules for atomic actions only).
- ActionDerivation: derived from inheritance hierarchy between Resources (ResourceDerivation). (Details technical and omitted.)

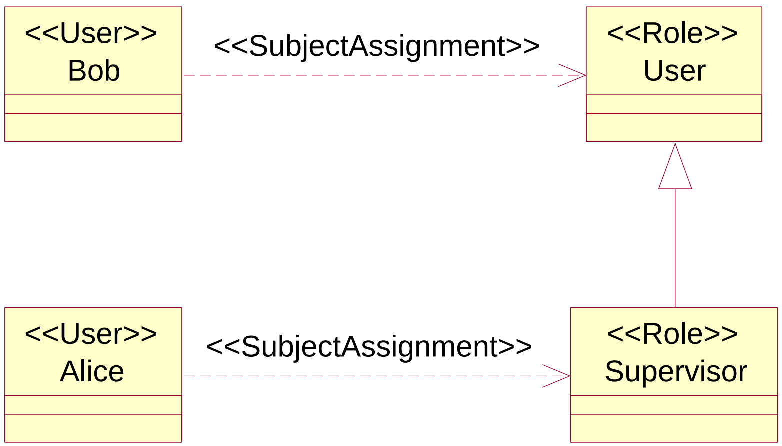
Authorization Constraints



- A permission can be restricted by an authorization constraint. E.g., user is account owner and amount is less than 1,000 EUR.
- This predicate describes an additional 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)

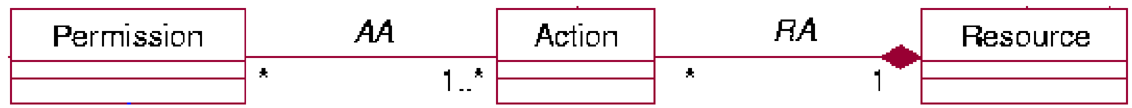
that must hold in order to grant access.

Roles and Users



- Concrete Syntax (UML-encoding) of SecureUML is defined by a UML-Profile (stereotypes, tagged values)
- Roles, Users, User-Role-Assignments and Role-Hierarchies are encoded as UML classes and relations with stereotypes.
- In practice, user administration and role assignments are not part of actual security model. These assignments are made after system deployment by system administrators.

Permissions



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

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

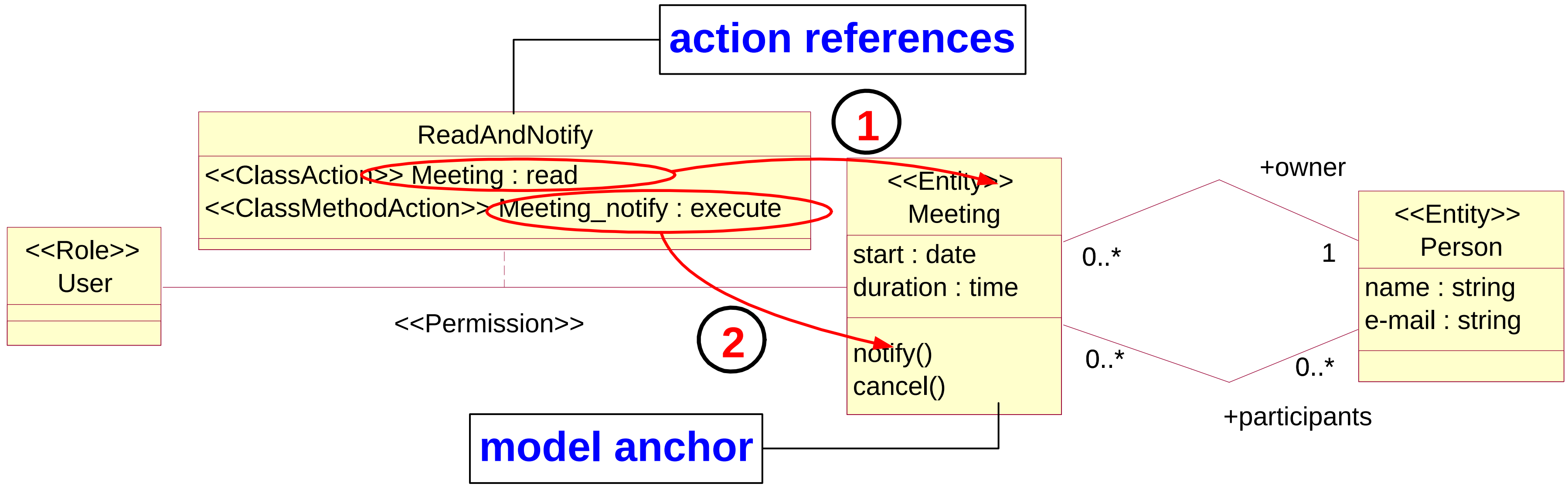
- A permission binds one or more actions to a single resource.

- Concrete syntax could directly reflect abstract syntax

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

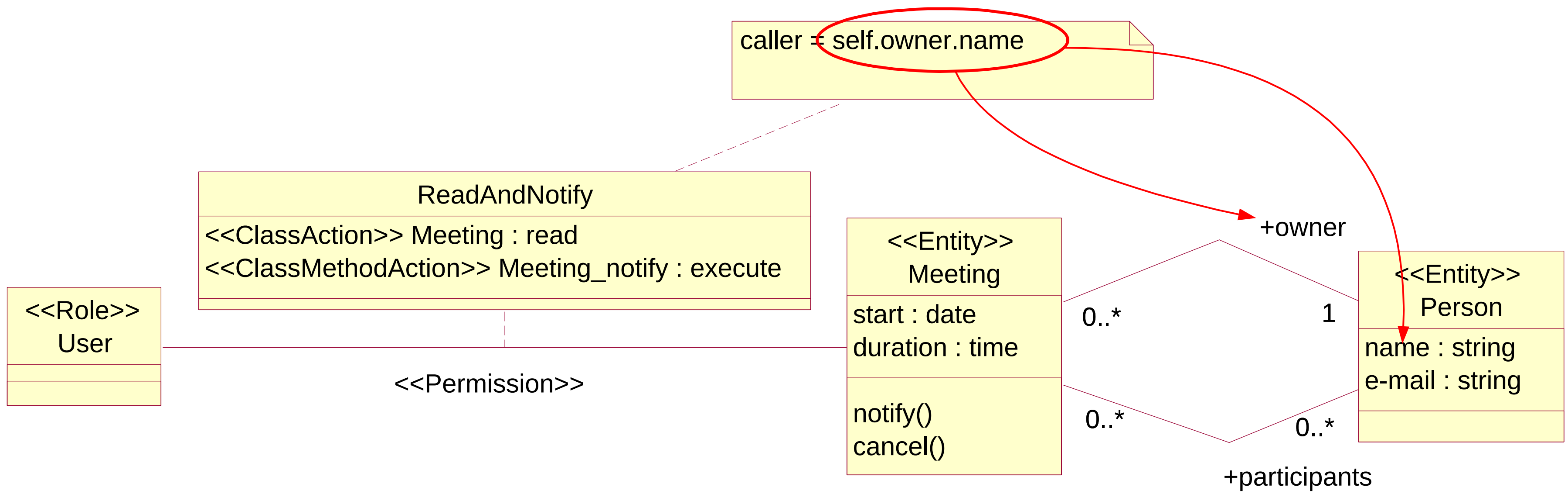
- Alternative: use association class to specify a ternary relation.
  - Attributes of association relate permissions with actions.
  - Actions identified by resource name and action name

Permissions (Cont.)



- Encoding as association class connecting a *role* and a *UML class* (model anchor).
- Actions of the model anchor (1) or its sub-elements (2) may be assigned to the permission (action references).

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

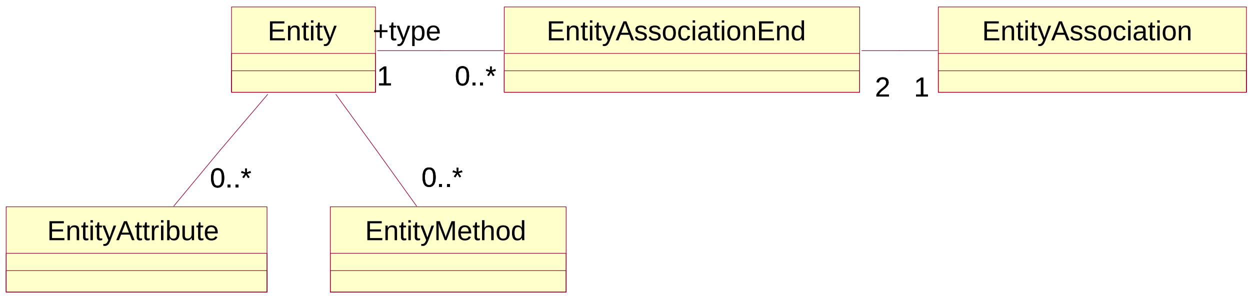
- Role-Based Access Control

- Generalization to SecureUML

☞ **Component modeling and combination**

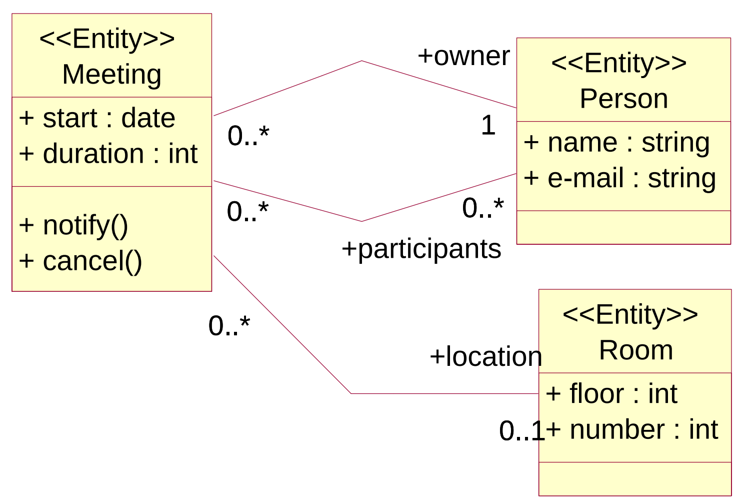
A Design Modeling Language for Components

- ComponentUML: a class based language for data modeling.

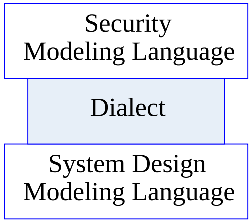


- Example design: group meeting administration system.

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».

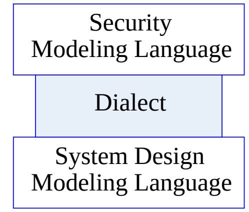
2. Identify protected resources

3. Identify resource actions

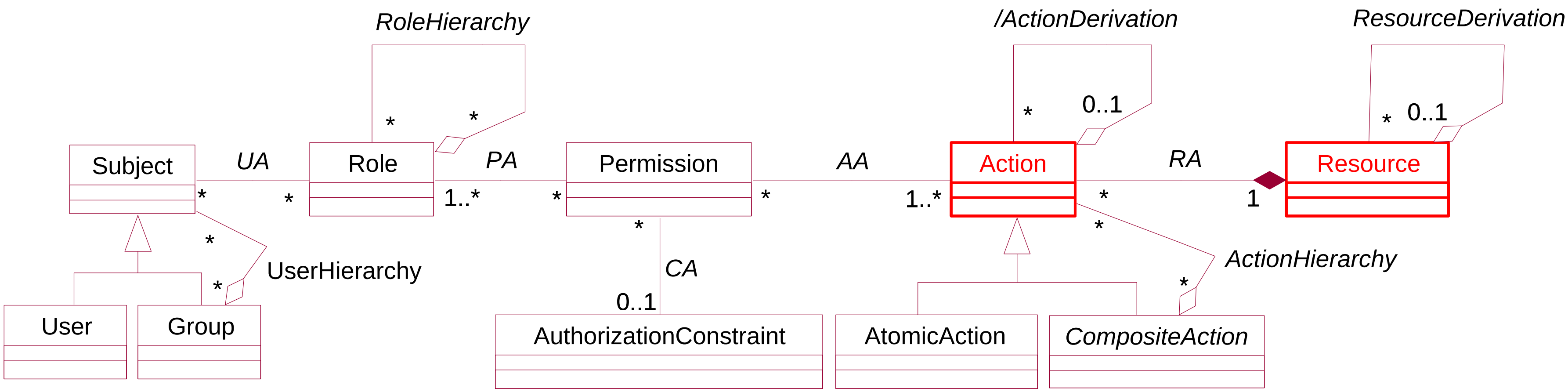
4. Define action hierarchy

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

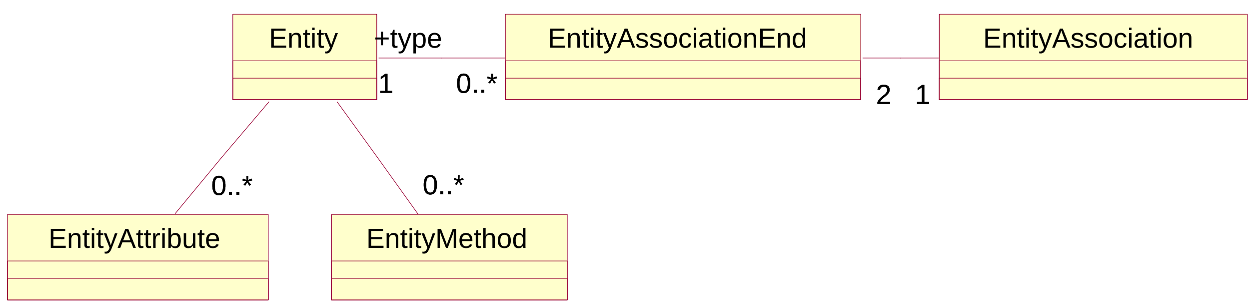
Defining a Dialect



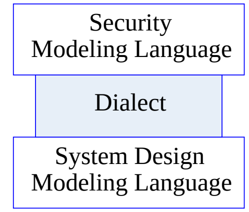
Security Modeling Language = SecureUML



System Design Modeling Language = Component UML

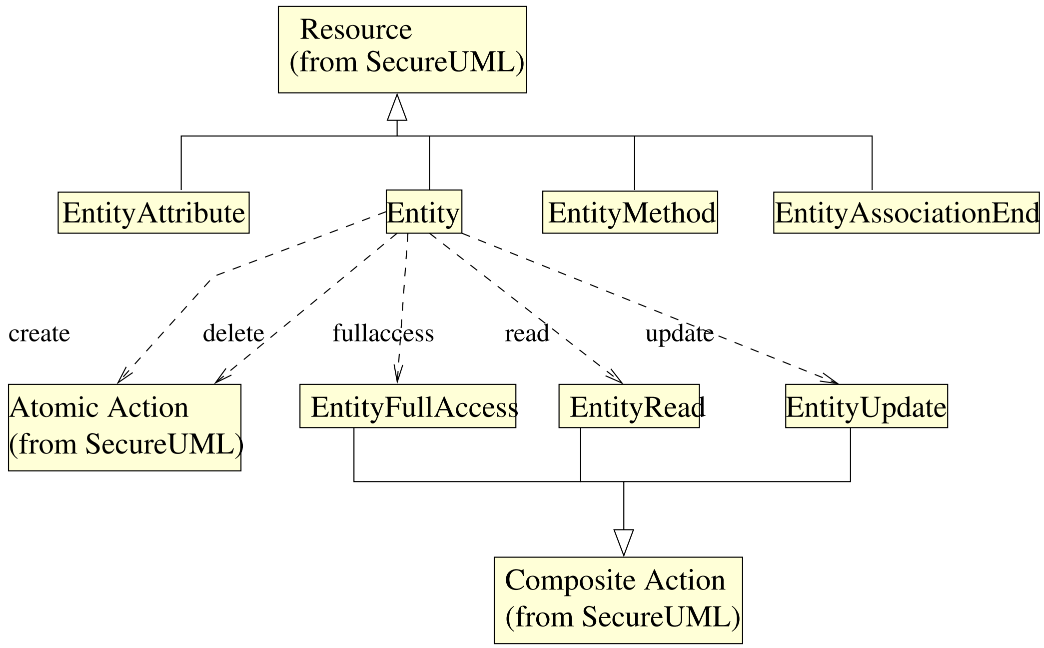


What are the resources and actions of ComponentUML?



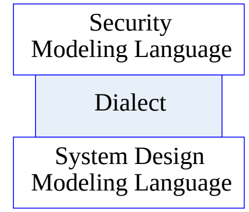
Defining a Dialect

- Identify protected resources and actions.



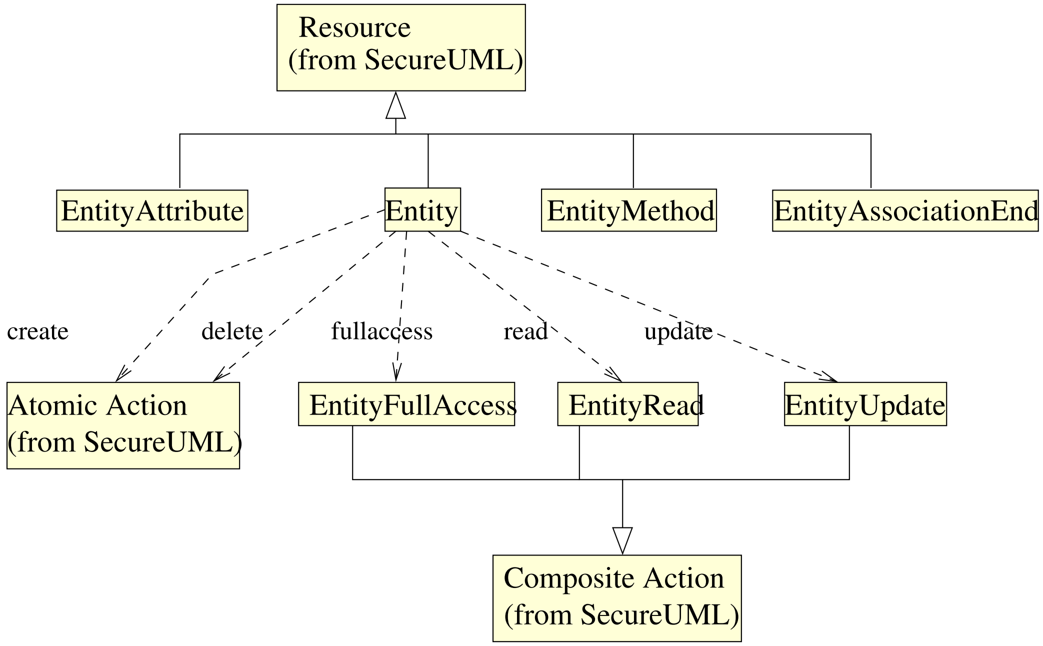
- As well as the action hierarchy (with blue atomic actions).

resource type	action	subordinated 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



Defining a Dialect

- Identify protected resources and actions.



- As well as the action hierarchy (with blue atomic actions).

resource type	action	subordinated 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

Defining a Dialect — Technical Details

- Resources are identified (graphically) using subtyping.

  Metatypes inherit from the SecureUML type Resource

- Resource actions are graphically defined using named dependencies from resource types to action classes (either atomic action or a subtype of composite action).

- Action hierarchy defined using OCL invariants

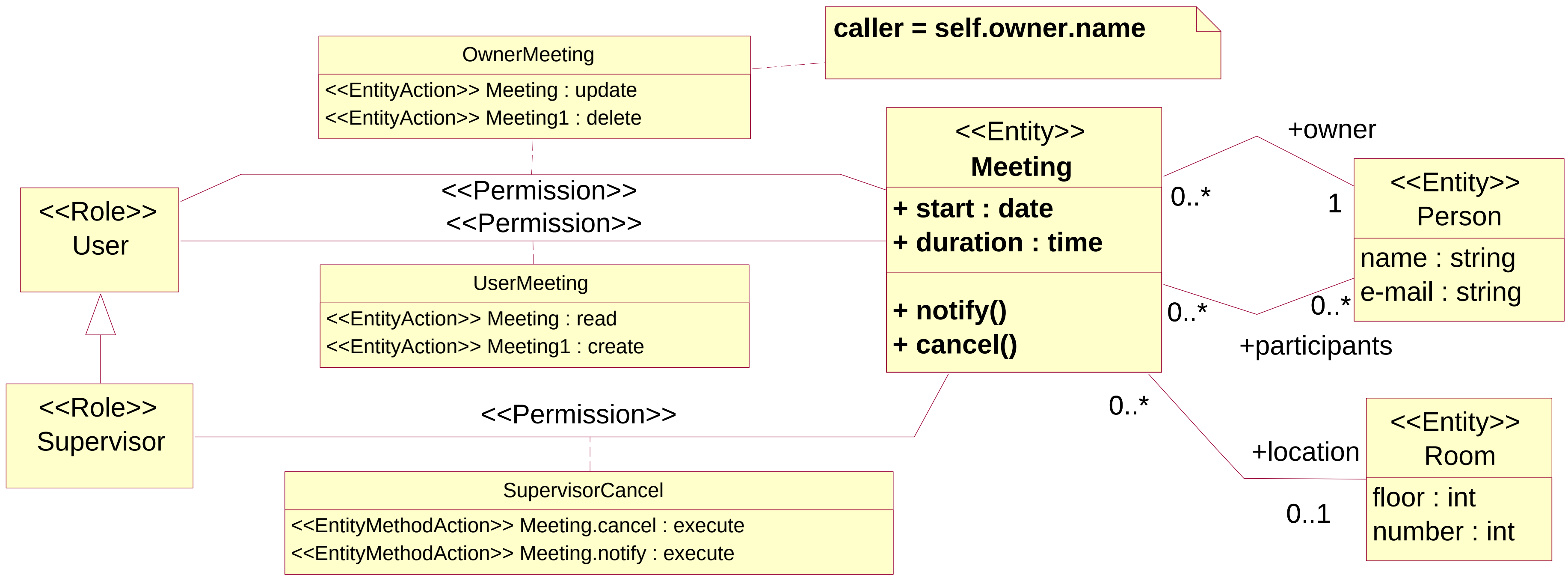
```
context EntityFullAccess inv:
subordinatedActions = resource.actions->select(
  name="create" or name="read" or name="update" or name="delete")
```

  Formalizes that the composite action *EntityFullAccess* is larger than the actions *create*, *read*, *update*, and *delete* of the entity the action belongs to.

Modeling a Security Policy

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.

Modeling a Security Policy



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.

Road Map

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

Semantics

☞ **SecureUML without constraints** (static, fixed at build time)

- Secure UML, adding constraints (state based)
- Semantics of general combinations (transition-system based)

Semantics: Why Bother?

Conceptually: what do all these boxes and arrows actually mean?

Note that a metamodel is not a model in the logical sense but rather a description of well-formed syntax.

Analysis: what are the consequences of what we have modelled?

Even when we understand all the modeling constructs, we may not understand all that our model entails.

Translation: are our generation functions correct?

Code has a semantics (at least an operational one).
Does it respect the model's semantics, in some appropriate sense?

General Idea

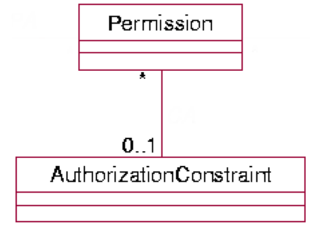
- SecureUML formalizes two kinds of Access Control decisions:
  1. Declarative Access Control, where decisions depend on static information: the assignments of users and permissions to roles.
  2. Programmatic Access Control, where decisions depend on dynamic information: the satisfaction of authorization constraints in the current system state.

- For (1), we cast the static (RBAC) information as a first-order structure $\mathfrak{S}_{RBAC}$. Semantics of declarative AC decisions given by

$$\mathfrak{S}_{RBAC} \models \phi_{RBAC}(u, a)$$

where $\phi_{RBAC}(u, a)$ formalizes that user u can perform action a.

Idea (cont.)



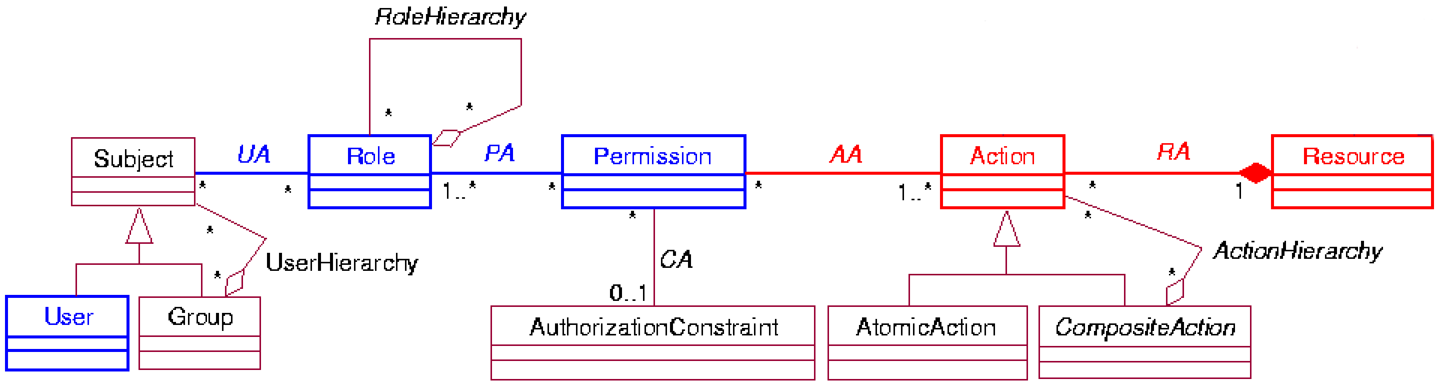
- (2) concerns conditions on permissions (as opposed to actions), whose satisfiability depends on system state.
  - system states $St \mapsto$ first-order structures $\mathfrak{S}_{St}$
  - authorization constraints $\mapsto$ formulas ϕ^p_{st} over states (ϕ^p_{st} denotes a family of formulae, one for each permission p)
  - Satisfiability of constraints in state $\mapsto \mathfrak{S}_{St} \models \phi^p_{st}$

- Combination interpreted by "combining" structures and formulas. Combined semantics roughly:

$$\langle \mathfrak{S}_{RBAC}, \mathfrak{S}_{St} \rangle \models \phi_{AC}(u, a)\,,$$

where $\phi_{AC}(u, a)$ is built from both $\phi_{RBAC}(u, a)$ and ϕ^p_{st}, and $\langle \mathfrak{S}_{RBAC}, \mathfrak{S}_{St} \rangle$ denotes the "union" of these structures.

Declarative Semantics



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

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

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

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

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

Declarative Semantics (without hierarchies)

- 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 and roles) $\mapsto$ tuples in the associated relation (e.g., UA).

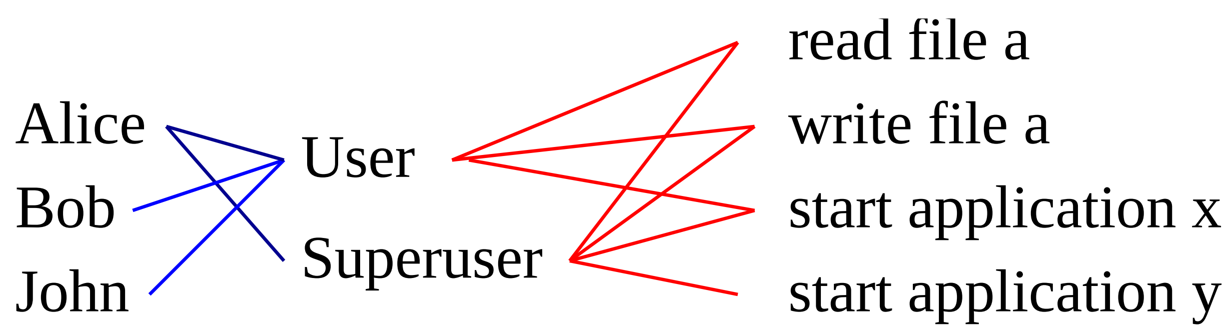
- $\phi_{RBAC}(u, a)$ formalizes standard RBAC semantics.
  - "Can user u perform permission p?"
    $\phi_{RBAC}(u, p) \iff (u, p) \in \mathsf{AC}$, where $\mathsf{AC} := \mathsf{PA} \circ \mathsf{UA}$.
  - is refined to: "Does user u have the permission to carry out action a?"
    $\phi_{RBAC}(u, a) \iff (u, a) \in \mathsf{AC}$, 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: $\mathfrak{S}_{RBAC} \models \phi_{RBAC}$.

AC Decision Problem: $\mathfrak{S}_{RBAC} \models \phi_{RBAC}$

- Problem is satisfiability in a finite structure, amounting to a graph.

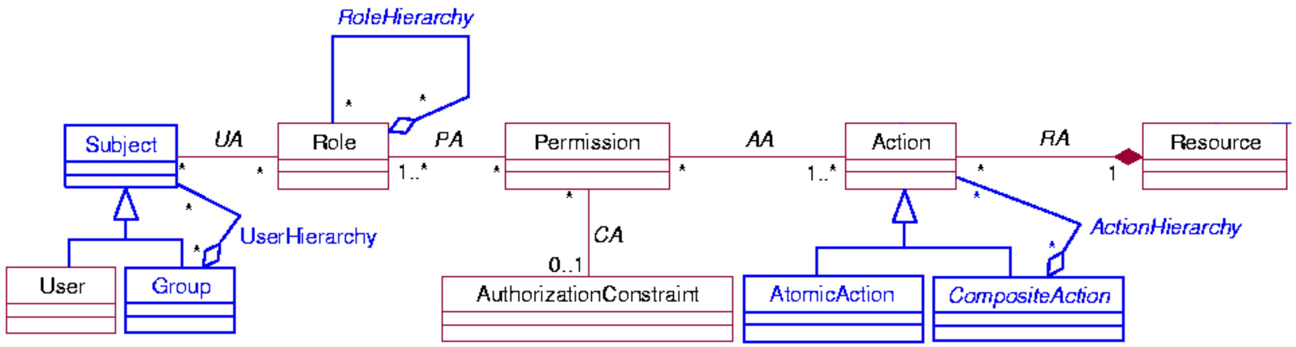


- Can John start application x?

  Just try all roles: complexity $O(|Roles|)$.

- When we add more sets and relations, depth first search can be used.

Adding Hierarchies



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

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

Declarative Semantics — Reformulation

$\phi_{RBAC}(u,a)$ with variables u of sort *Users* and a of sort *Actions* is defined by

$$\exists s \in \mathit{Subjects}, r_1, r_2 \in \mathit{Roles}, p \in \mathit{Permissions}, a' \in \mathit{Actions}.$$
$$s \geq_{\mathit{Subjects}} u \wedge UA(s, r_1) \wedge r_1 \geq_{\mathit{Roles}} r_2 \wedge$$
$$PA(r_2, p) \wedge AA(p, a') \wedge a' \geq_{\mathit{Actions}} a \ ,$$

This can be equivalently formulated by factoring out the permissions explicitly:

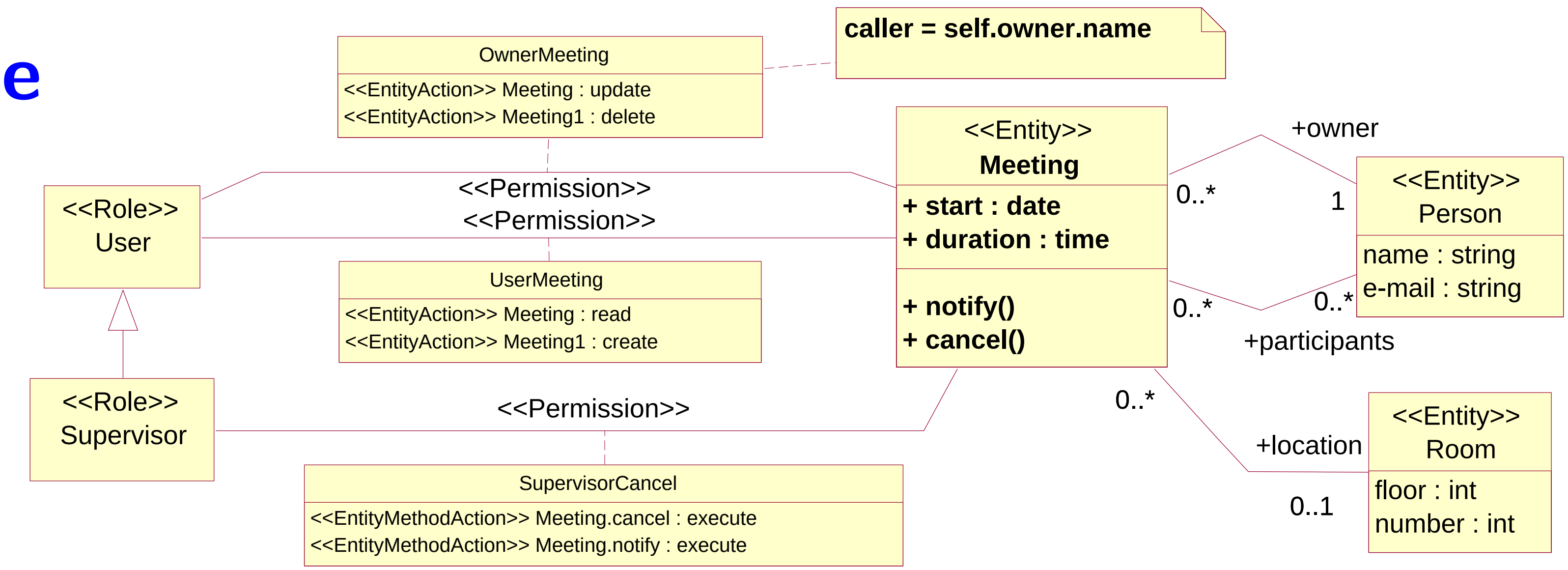
$$\phi_{RBAC}(u,a) = \bigvee_{\{p \in Permissions\}} \phi_{\mathit{User}}(u,p) \wedge \phi_{\mathit{Action}}(p,a) \ ,$$

where

$$\phi_{\mathit{User}}(u,p) \equiv (u,p) \in \mathsf{PA} \circ \geq_{Roles} \circ \, \mathsf{UA} \circ \leq_{Subjects}$$

$$\phi_{\mathit{Action}}(p,a) \equiv (p,a) \in \geq_{Actions} \circ \, \mathsf{AA}$$

Example



Assume configuration with users Alice and Bob where Alice is a Supervisor.

$$
\begin{aligned}
Users &= \{\text{Alice}, \text{Bob}\} \\
Roles &= \{\texttt{User}, \texttt{Supervisor}\} \\
Permissions &= \{\texttt{OwnerMeeting}, \texttt{UserMeeting}, \texttt{SupervisorCancel}, \ldots\} \\
Actions &= \{\texttt{Meeting.update}, \texttt{Meeting::cancel.execute}, \ldots\} \\
UA &= \{(\text{Bob}, \texttt{User}), (\text{Alice}, \texttt{Supervisor})\} \\
PA &= \{(\texttt{User}, \texttt{OwnerMeeting}), (\texttt{Supervisor}, \texttt{SupervisorCancel}), \ldots\} \\
AA &= \{(\texttt{OwnerMeeting}, \texttt{Meeting.update}), (\texttt{SupervisorCancel}, \texttt{Meeting::cancel.execute}), \ldots\} \\
\geq_{Roles} &= \{(\texttt{Supervisor}, \texttt{User}), (\texttt{Supervisor}, \texttt{Supervisor}), (\texttt{User}, \texttt{User})\} \\
\geq_{Actions} &= \{(\texttt{Meeting.update}, \texttt{Meeting::cancel.execute}), \ldots\} \ ,
\end{aligned}
$$

Semantics

- SecureUML without constraints (static, fixed at build time)

☞ **Secure UML, adding constraints** (state based)

- Semantics of general combinations (transition system based)

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

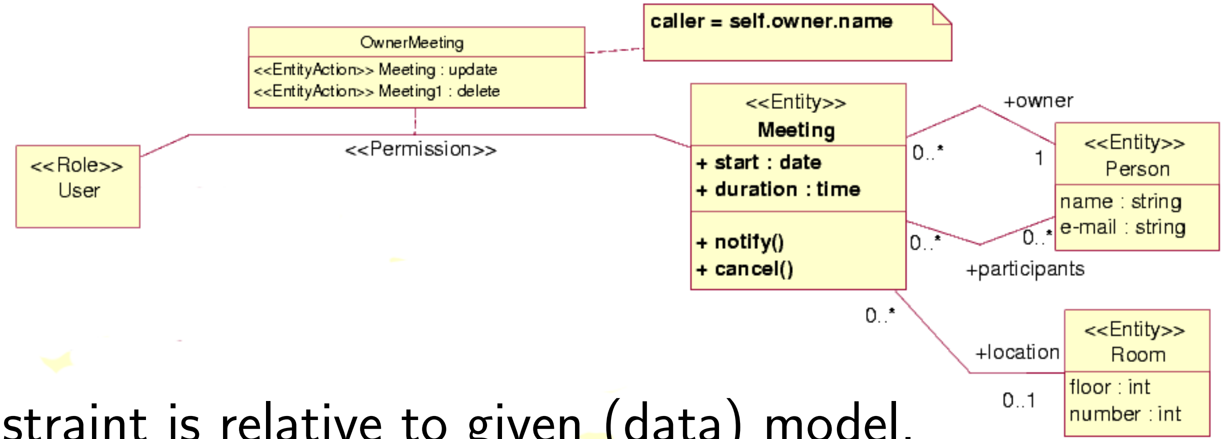
- Straightforward translation into sorted FOL, e.g.,

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

- Semantics of OCL relative to a system state (or “snapshot”).
  - Can be recast as a structure, giving semantics for translation.
  - Fixes objects, their attribute values, and which pairs of objects are instances of associations.

Details, e.g., Beckert, Keller, Schmitt, *Translating the Object Constraint Language into First-order Predicate Logic, 2002.*

Constraint Syntax



- Vocabulary of OCL constraint is relative to given (data) model.

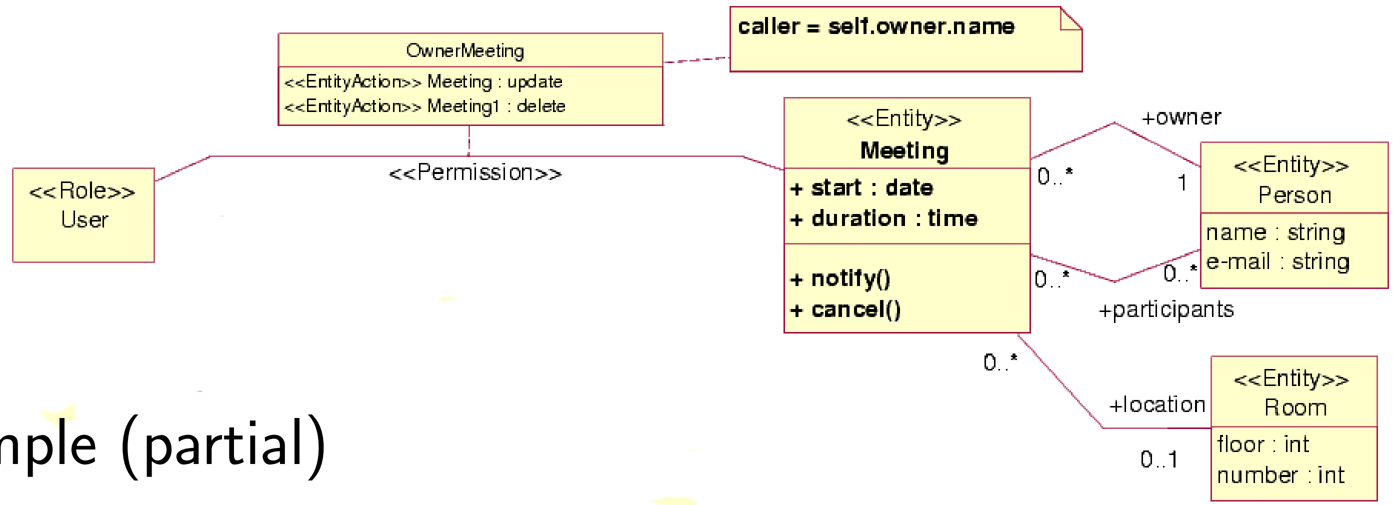
  Same holds for translation ϕ^p_{st}: model determines Σ_{St}.

- Let's consider this for class diagrams. $\Sigma_{St} = (S, F, P)$ contains:

  $\mathcal{S}_{St}$: 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.

- And additionally
  - $\mathcal{F}_{St}$ contains constant *self*$_C$ for each class C in the system model.
  - Sorts, functions, and predicates over base types like $Integer$ and $String$.
  - A constant symbol $caller$ of type $String$ denoting the name of the user on whose behalf an action is performed at a time point t.

Example



- Signature for example (partial)

$$\begin{aligned} S &:= \{\mathit{Meeting}, \mathit{Person}, \ldots\} \cup \{\texttt{String}, \ldots\} \\ F &:= \{\mathit{self}_{\mathit{Meeting}}, \ldots, \mathit{meetingOwner}, \mathit{personName}\} \cup \{\texttt{caller}\} \\ P &:= \{\mathit{meetingParticipants}, \ldots\} \cup \{=_{String}, \ldots\} \end{aligned}$$

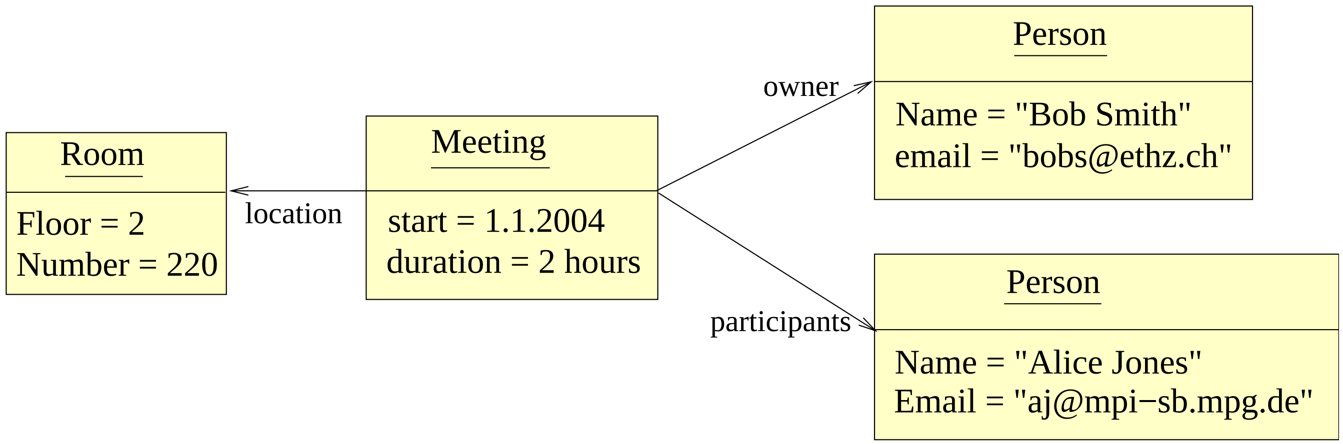
- $\phi_{St}^{OwnerMeeting}$ is:

$$\texttt{caller} \quad =_{String} \quad \mathit{personName}(\mathit{meetingOwner}(\mathit{self}_{\mathit{Meeting}}()))$$

Formalizes that the method caller's (authenticated) name is the same as the name of the person who is the owner of the meeting.

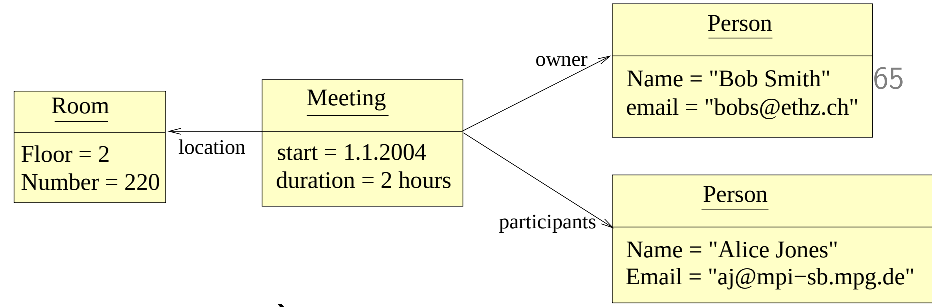
Constraint Semantics

- A system snapshot at any point 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
  - Attributes and references define functions (or relations) from objects to corresponding values.
  - Currently executing object of class C gives interpretation for self_C.

Snapshot $\mapsto \mathfrak{S}_{St}$



- Object of class **C** $\mapsto$ element of (carrier set of) sort C.
- Attribute in class **C** of type **T** $\mapsto$ function of type $C \to T$, returning the attribute value of object to which it is applied.

  E.g., $\mathit{Name} : \mathit{Person} \to \mathit{String}$ returns value of *Name* attribute of person object in current state.
- Side-effect-free methods and attribute ends handled analogously.
- Base types/functions/predicates have their standard interpretations.
- $\mathit{self}_C \mapsto$ the currently executing object (of type C).
  (Fix an arbitrary interpretation of self_D for all D, $D \neq C$.)
- *caller* $\mapsto$ name of the (authenticated) user executing the method.
- A constraint ϕ_{St} is satisfied iff $\mathfrak{S}_{St} \models \phi_{St}$.

RBAC + Constraints

- A user u can perform an action a if he has a permission for this action where the associated constraint is satisfied:

$$\phi_{AC}(u,a) = \bigvee_{p \in Permissions} \phi_{\mathsf{User}}(u,p) \wedge \phi_{\mathsf{Action}}(p,a) \wedge \phi_{st}^{p}$$

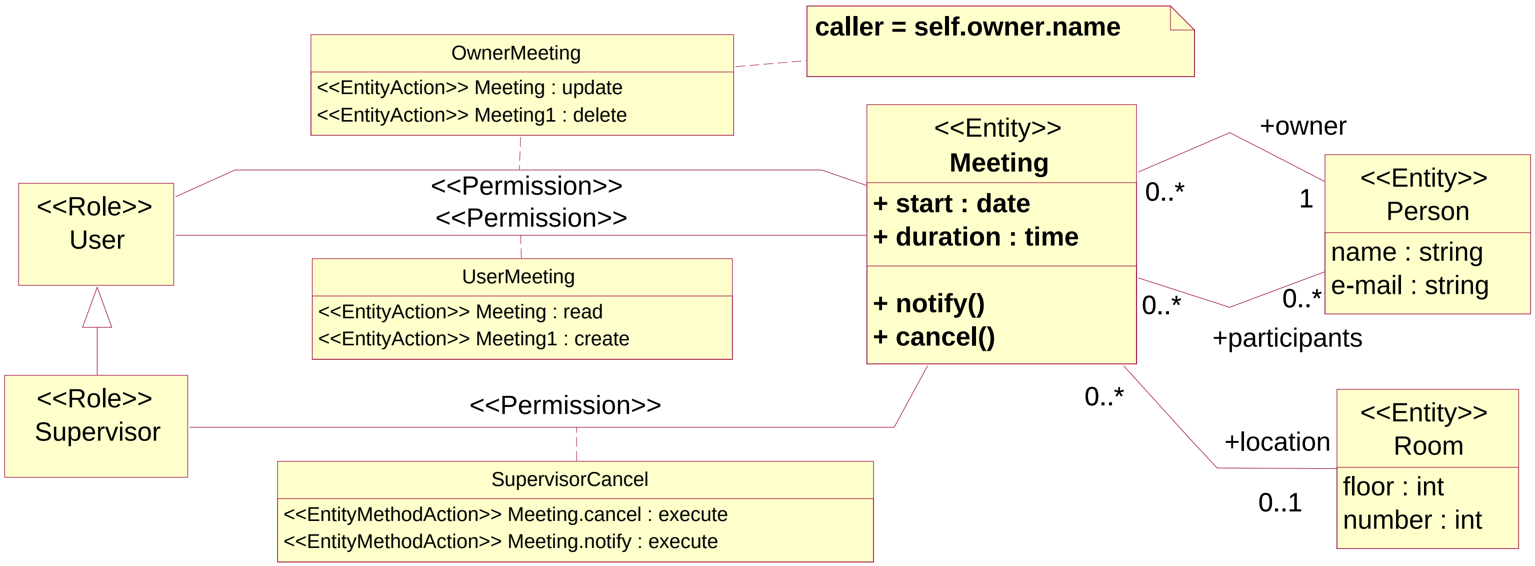
- Formulae ϕ_{AC} are built over a signature combining Σ_{RBAC} and Σ_{St} by taking their union (unproblematic as signatures disjoint).

$$\Sigma_{AC} = (\mathcal{S}_{RBAC} \cup \mathcal{S}_{St}, \mathcal{F}_{RBAC} \cup \mathcal{F}_{St}, \mathcal{P}_{RBAC} \cup \mathcal{P}_{St})$$

- $\mathfrak{S}_{AC} = \langle \mathfrak{S}_{RBAC}, \mathfrak{S}_{St} \rangle$ is the structure that consists of the carriers sets, functions and predicates from both $\mathfrak{S}_{RBAC}$ and $\mathfrak{S}_{St}$.

- AC decision: $\mathfrak{S}_{AC} \models \phi_{AC}$.

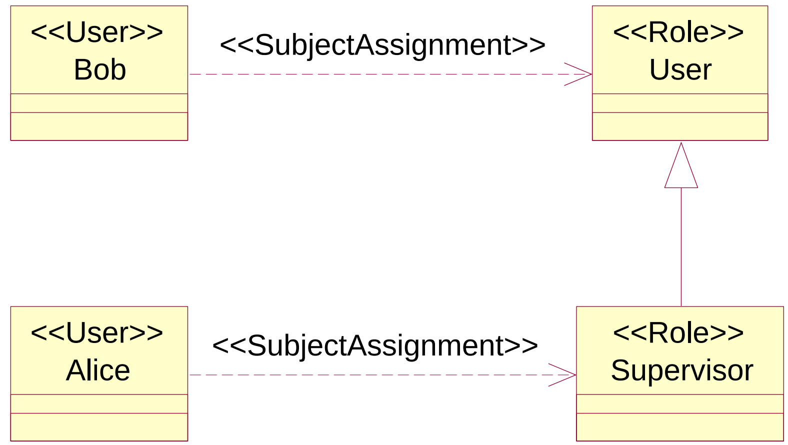
Example Again



$$
\begin{aligned}
\text{Roles} &:= \{\texttt{User}, \texttt{Supervisor}\} \\
\text{Permissions} &:= \{\texttt{OwnerMeeting}, \texttt{UserMeeting}, \texttt{SupervisorCancel}\} \\
\text{AA} &:= \{(\texttt{OwnerMeeting}, \texttt{Meeting.update}), (\texttt{SupervisorMeeting}, \texttt{Meeting.cancel.execute}), \ldots\} \\
\text{PA} &:= \{(\texttt{User}, \texttt{OwnerMeeting}), (\texttt{Supervisor}, \texttt{SupervisorCancel}), \ldots\} \\
\text{CA} &:= \{(\texttt{OwnerMeeting}, \texttt{"caller = self.owner.name"})\} \\
\geq_{Roles} &:= \{(\texttt{Supervisor}, \texttt{User})\}
\end{aligned}
$$

Example (cont.)

Assigning Bob to the role User
and Alice to the role Supervisor



means our structure contains

$$\begin{aligned} \text{Users} &:= \{\text{Bob}, \text{Alice}\} \\ \text{UA} &:= \{(\text{Bob}, \texttt{User}), (\text{Alice}, \texttt{Supervisor})\} \\ \geq_{Roles} &:= \{(\texttt{Supervisor}, \texttt{User})\} \end{aligned}$$

Recall $\phi_{AC}(u, a)$: $\bigvee_{p \in Permissions} \phi_{\mathsf{User}}(u, p) \wedge \phi_{\mathsf{Action}}(p, a) \wedge \phi^{p}_{st}$

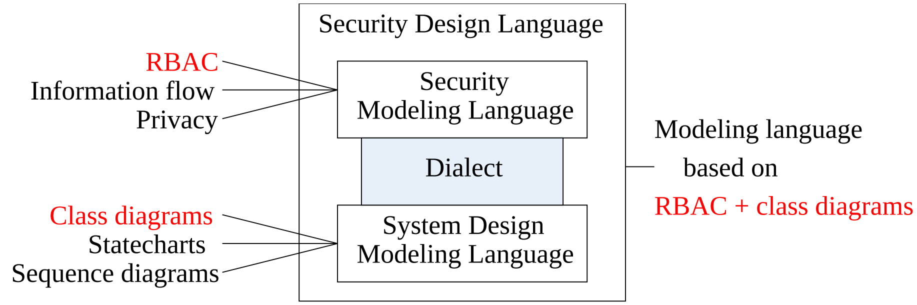
So in this example, Alice could execute the action Meeting.update, in any state where $\texttt{caller} = personName(meetingOwner(\mathsf{self}_{\mathsf{Meeting}}))$ is satisfied, i.e., where she is the meeting's owner.

Semantics

- SecureUML without constraints (static, fixed at build time)
- Secure UML, adding constraints (state based)

☞ **Semantics of general combinations** (transition system based)

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 semantics of the combination?

  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.

  Namely, semantics must be expressible as an LTS.

Semantics of Design Modeling Language

- 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:

  $s_0 \stackrel{a_0}{\rightarrow} s_1 \stackrel{a_1}{\rightarrow} \dots \stackrel{a_n}{\rightarrow} s_{n+1}$ is a possible behavior iff $(s_i, a_i, s_{i+1}) \in \delta$, for $0 \le i \le n$.
- Combination with SecureUML yields LTS $\Delta_{AC} = (Q_{AC}, A_{AC}, \delta_{AC})$.

  Traces of Δ_{AC} should be a subset of those of Δ, where just those traces with prohibited actions are removed.

From Δ to $\Delta_{AC} = (Q_{AC}, A_{AC}, \delta_{AC})$

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

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

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

- δ_{AC} restricts δ and lifts to Q_{AC}:

$$\delta_{AC} = \{(q, a, q') \in \mathit{lift}(\delta) \mid q \models \phi_{AC}(u, a)\} \ ,$$

  where *lift*(δ) denotes the lifting of δ to Q_{AC}, i.e.,

$$\mathit{lift}(\delta) = \left\{ (q, a, q') \in Q_{AC} \times A_{AC} \times Q_{AC} \mid \begin{array}{l} (\pi_{St}(q), a, \pi_{St}(q')) \in \delta \wedge \\ \pi_{RBAC}(q) = \pi_{RBAC}(q') \end{array} \right\} \ ,$$

  and $\pi_{St} : Q_{AC} \to Q$ and $\pi_{RBAC} : Q_{AC} \to Q_{RBAC}$ are projections.

- N.B.: RBAC configuration never changes, i.e., it really is static.

Example: SecureUML + ComponentUML

- ComponentUML as 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.,

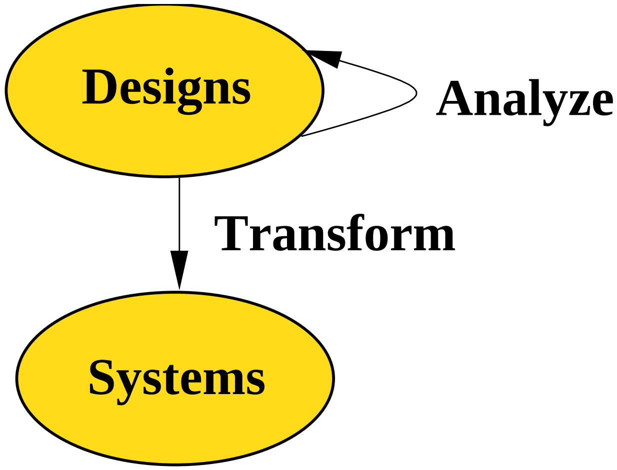
$$(set_{at}, e, v) \in \bigcup_{\{at \in Attributes\}} \{set_{at}\} \times Q_e \times Q_{at} \, ,$$

    where Q_e and Q_{at} denote the sets of all possible instances of the type of the attribute's entity, and the type of the attribute respectively.
  - ▶ δ defined by semantics of methods themselves. E.g.,
    - ∗ if a is a "get" action $(q, a, q') \in \delta$ iff $q = q'$.
    - ∗ if $a = (set_{at}, e, v)$ is a "set" action, then $(q, a, q') \in \delta$ implies $q' \models get_{at}(e) = v$.

- Combined semantics $\Delta_{AC} = (Q_{AC}, A_{AC}, \delta_{AC})$ as just described.

  δ_{AC}, contains only transitions allowed by SecureUML semantics.

Semantics: What's It Good For?



Analysis: Answer questions like:

- Is a given trace possible?
- Can Alice reach a given state?
- Which users may reach that state?

Current work is on model checking: Semantics can be translated into a rewriting logic theory and Maude tools used to answer such questions.

Correctness:

$$\mathfrak{S}_{AC} \models_{\text{SecureUML}} \phi_{AC}(u,p) \iff \text{“Implementation”} \models_{\text{EJB}} \phi_{AC}(u,p).$$

- Semantics provides basis for judging correctness of translation.
- For high-level pen-paper verification, see course notes.

Road Map

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

Generating Security Infrastructures

☞ **Generating EJB Infrastructures.**

- Motivation
- Basics of EJB and EJB access control
- Generation Rules
- Correctness

- Generating .NET Infrastructures.

Why Transform?

Decreases burden on programmer.

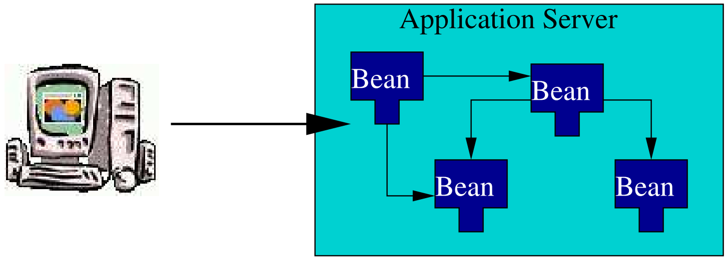
Faster adaption to changing requirements.

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.

Basic EJB concepts



- Enterprise Java Beans (EJB) is a widely used component architecture.
- Components (Beans) are executed inside of an application server.
- The server (container) is responsible for
  - persistency, authentication, transaction management,...
  - access control

EJB: Deeclarative and Programmatic AC

- Static configurations mapped to deployment descriptors, which are controlled by the application server.
  - The protected resources are the components' methods.
  - Mapping not direct. EJB supports vanilla RBAC without hierarchies.
- Runtime evaluation of assertions embedded in the application code, with mechanisms to access security-relevant data of the current user, e.g., his name or his roles.

```
<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>
```

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

Starting Points of Transformation

- Basis: Existing (MDA-style) transformation functions that transform ComponentUML models into EJB applications.

- Objective 1: Adhere to the SecureUML semantics:

$$\phi_{AC}(u,a) = \bigvee_{p \in Permissions} \phi_{\mathit{User}}(u,p) \wedge \phi_{\mathit{Action}}(p,a) \wedge \phi^{p}_{st}$$

$$\phi_{\mathit{User}}(u,p) \equiv (u,p) \in \mathsf{PA} \circ \geq_{Roles} \circ \mathsf{UA} \circ \leq_{Subjects}$$

$$\phi_{\mathit{Action}}(p,a) \equiv (p,a) \in \geq_{Actions} \circ \mathsf{AA}$$

- Objective 2: Use what is available on the technology platform: EJB supports both declarative RBAC (without role-hierarchies) and runtime access to security-relevant data of the current user.

Overview of Transformation Rules

- Permissions for atomic actions.
  - Generate `method-permission` elements of the corresponding method in the deployment descriptor, naming all authorized roles.
  - Calculate these roles according to the hierarchy on action ($\geq_{\mathsf{Actions}}$), the assignment of actions to permissions (AA), the assignment of permissions to roles (PA), and the hierarchy on roles $\geq_{\mathsf{Roles}}$.

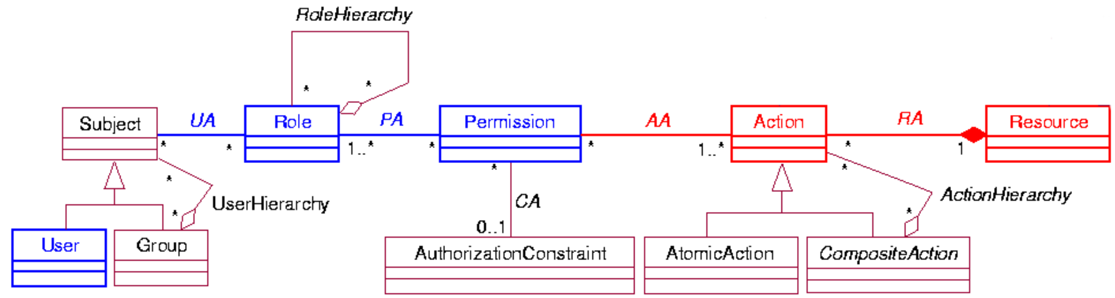
- Permissions for composite actions.

  No rules needed: action hierarchy is used when calculating roles above.

- Authorization constraints on permissions.

  Add assertions at the start of the corresponding methods, checking the necessary roles and evaluating the constraint(s).

Transformation Rules Static Part



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:

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

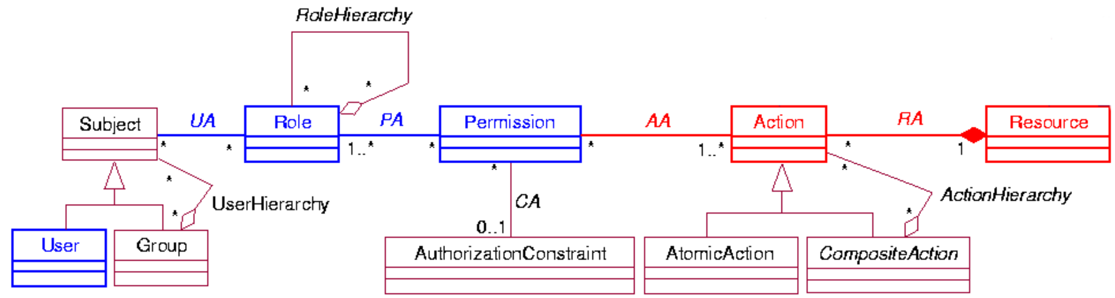
This can be done by (depth-first) searching the directed acyclic graph defined by the relations AA, $\geq_{\text{Actions}}$, PA, $\geq_{\text{Roles}}$.

☞ generate the following 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>
```

David Basin

Transformation Rules Dynamic Part



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.

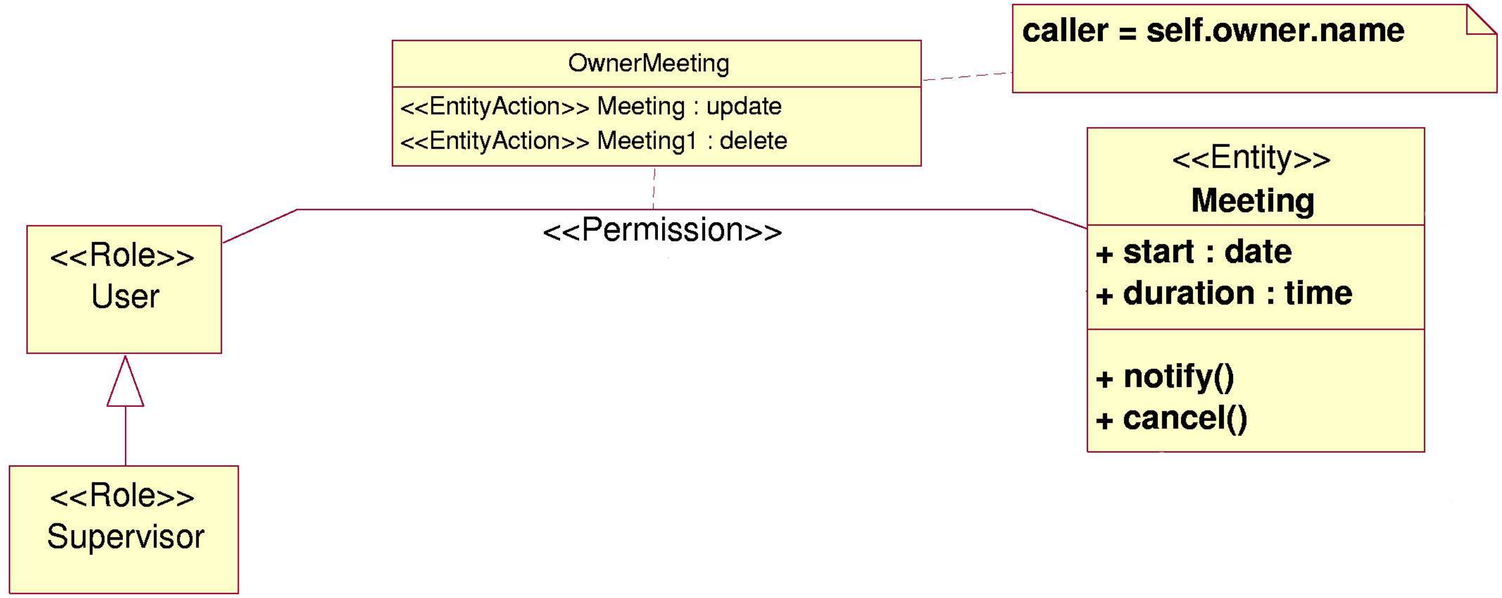
☞ if yes, include at the start of the method m the assertion:

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

```
throw new AccessControlException("Access denied."); ,
```

where Constraint(p) is the translation of the attached constraint into 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.");
...
}
```

Generating Security Infrastructures

- Generating EJB Infrastructures.

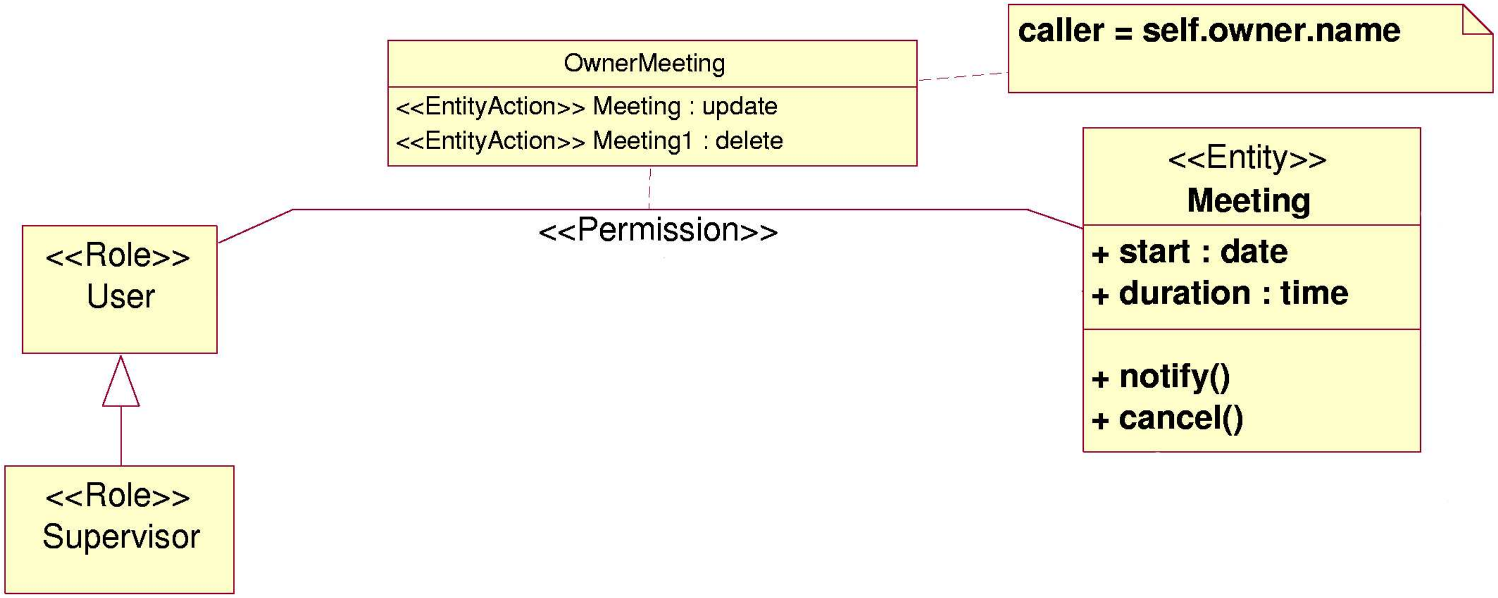
☞ **Generating .NET Infrastructures.**

.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

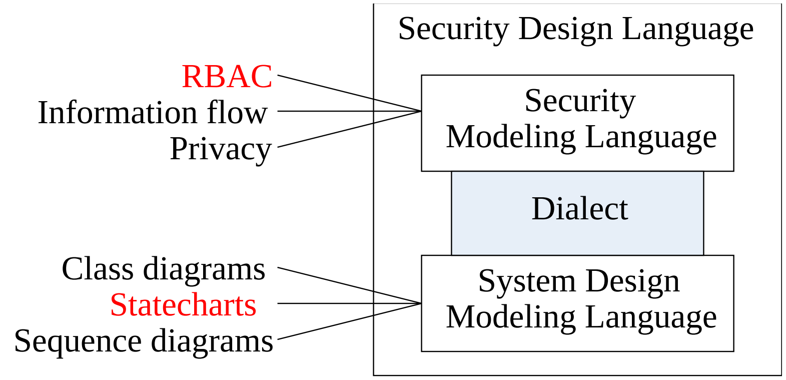
- Motivation and objectives
- Background
- Secure components
- Semantics
- Generating security infrastructures
- ☞ **Secure controllers**
- Experience, demonstration, and conclusions

Secure Controllers

☞ **ControllerUML: a modeling language for controllers.**

- Integrating ControllerUML with SecureUML.
- Generating secure web applications based on the Java Servlet architecture.

Motivation



- Explore parameter space.
  Integrate SecureUML with a process-oriented modeling language.

- Applications include:
  - Work-flow management: Restrict process execution to entitled parties.
  - Application controllers: Build a first line of defense against attackers in multi-tier systems.

  Control access to states, transitions, associated actions, etc.

- Explore interrelationships between different views.
  E.g., securing controllers versus securing components versus ...

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
  - A user-interface of an application changes its state according to clicks on certain menu-entries.
  - 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.

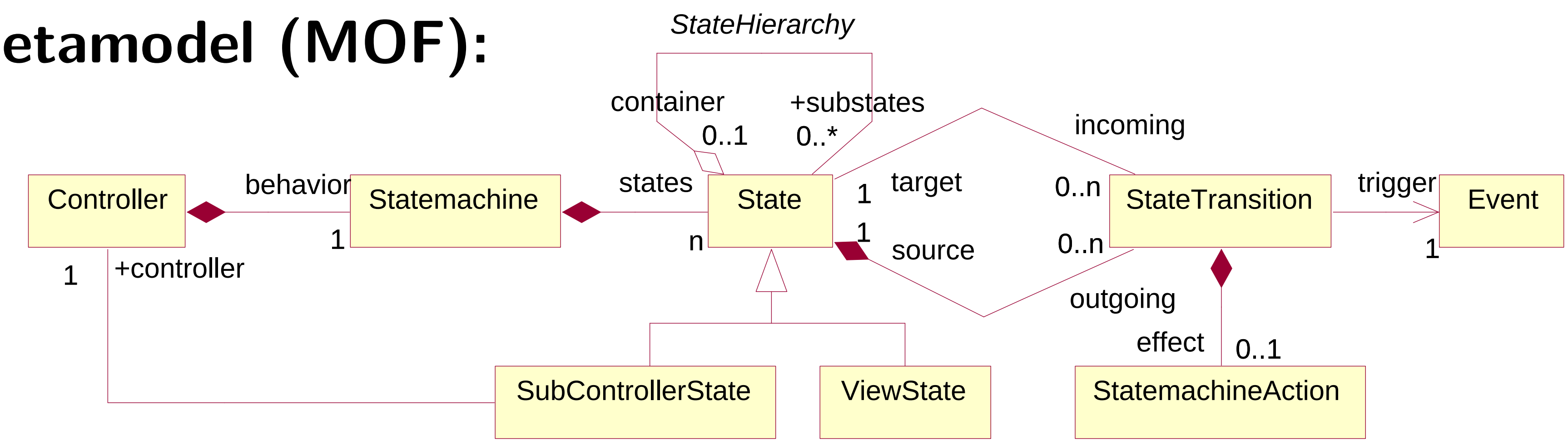
  Persistence tier: where data (model) is stored, e.g., backend data-base system.

  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.

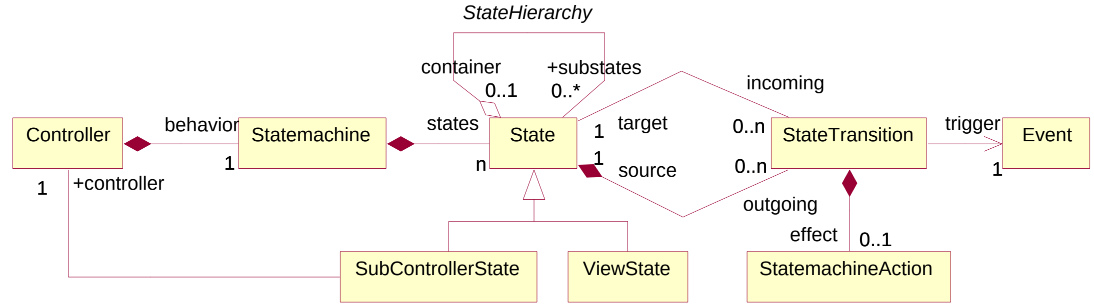
Abstract Syntax

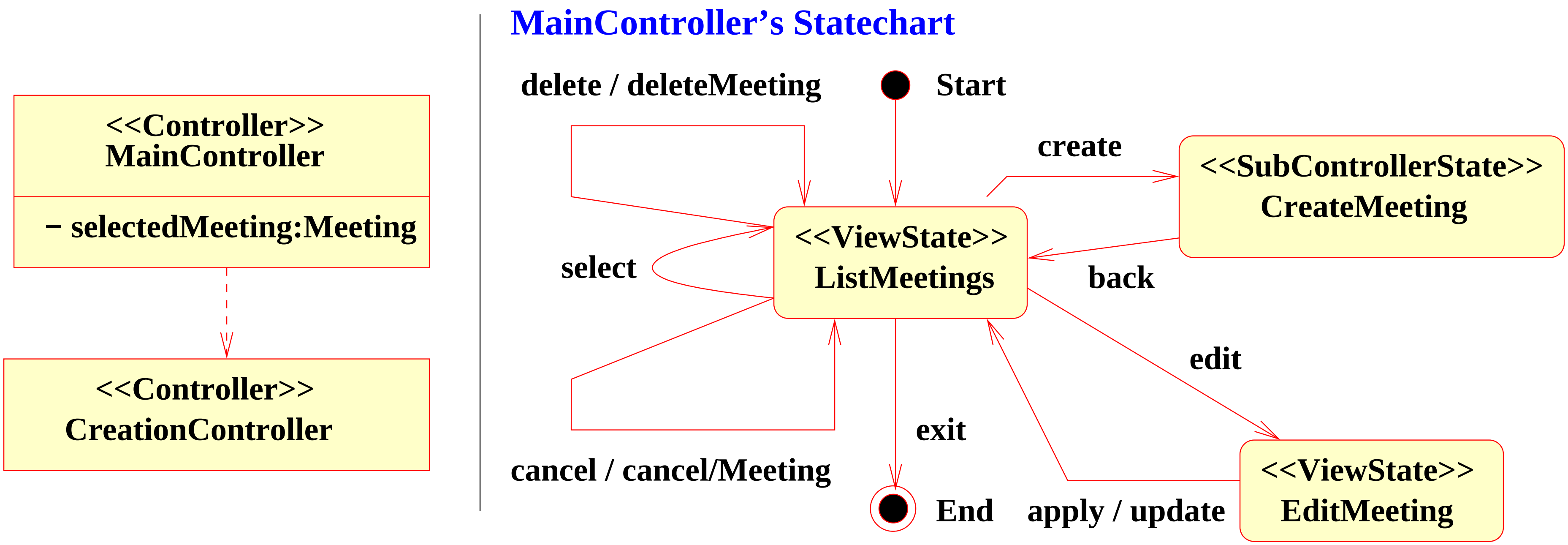
Metamodel (MOF):



- 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.

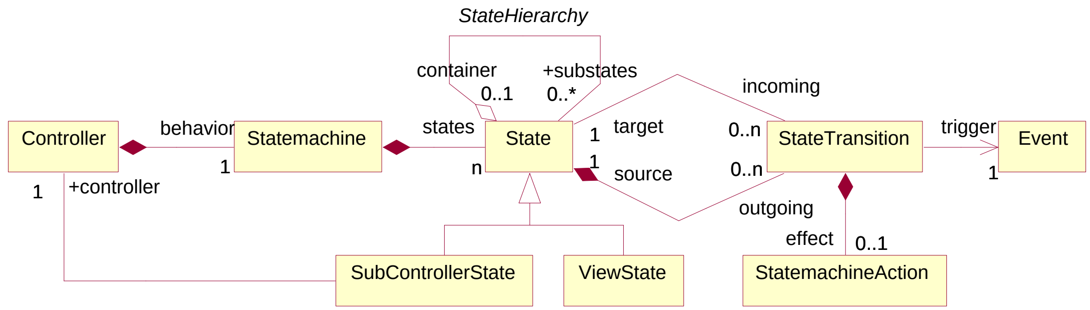
Concrete Syntax

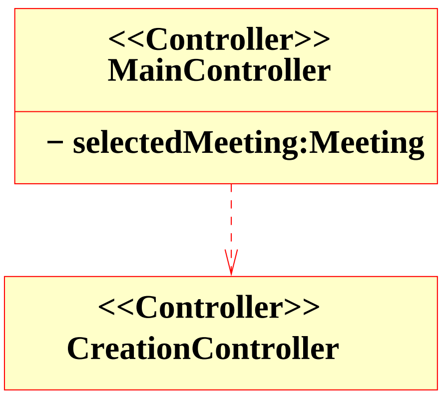




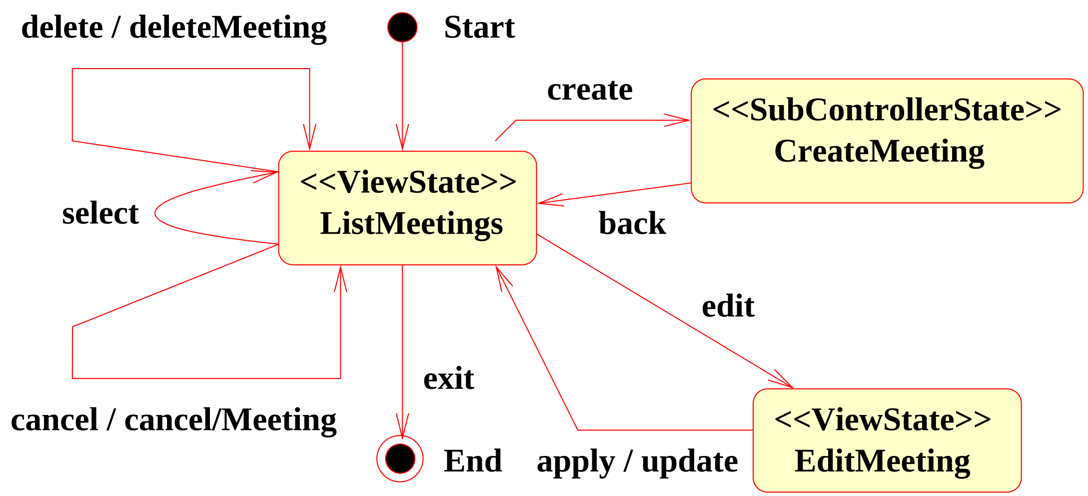
- Concrete syntax defined by a UML-profile (stereotypes, tagged values).
- Encoding uses elements from both UML class diagrams and state charts. (References not visualized, e.g., from subcontroller states to controllers are stored in tagged values.)

Syntax (cont.)



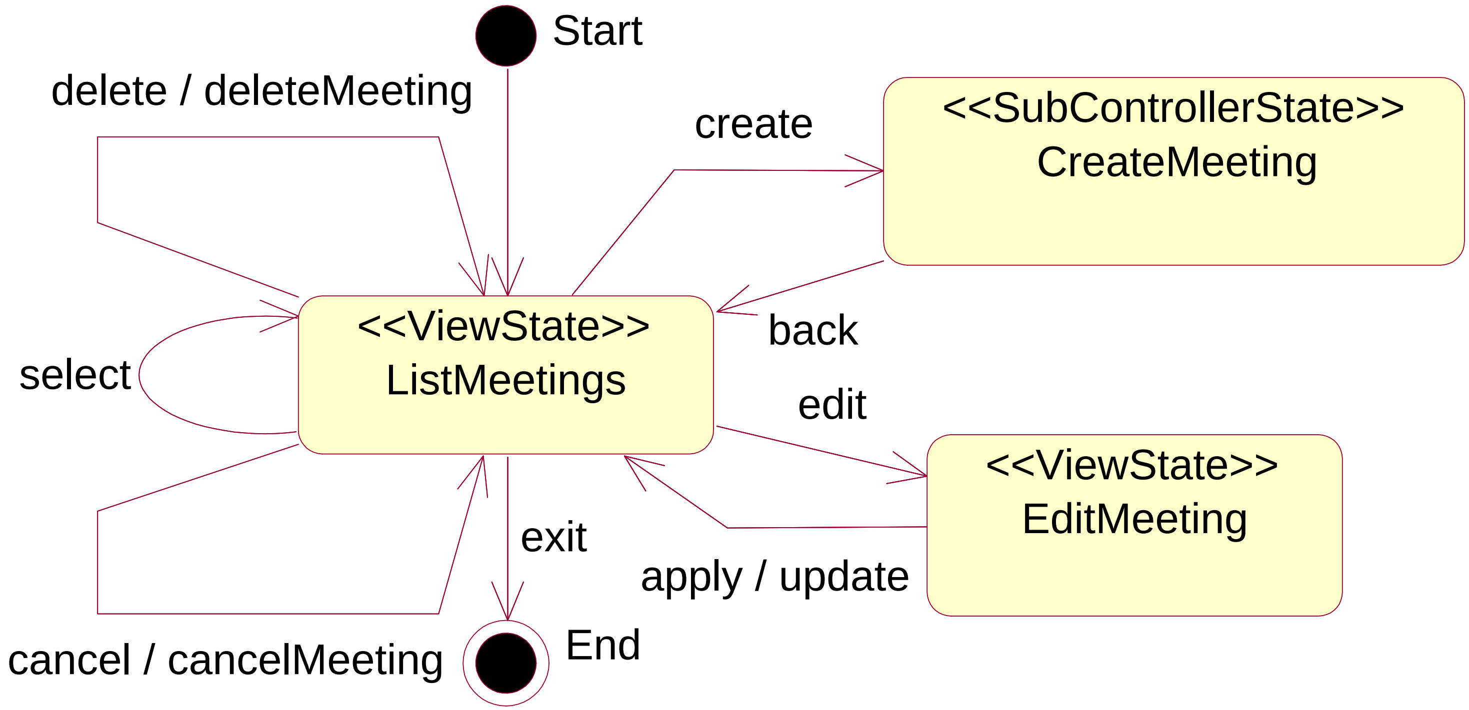


MainController's Statechart



- Controller $\mapsto$ UML class (stereotype "Controller") with an assigned UML statemachine.
- State, Transition, Event, and StatemachineAction $\mapsto$ their respective UML counterparts (transition name := name of triggering event).
- ViewState/SubControllerState $\mapsto$ UML state with corresponding stereotype.

Statemachine of the MainController

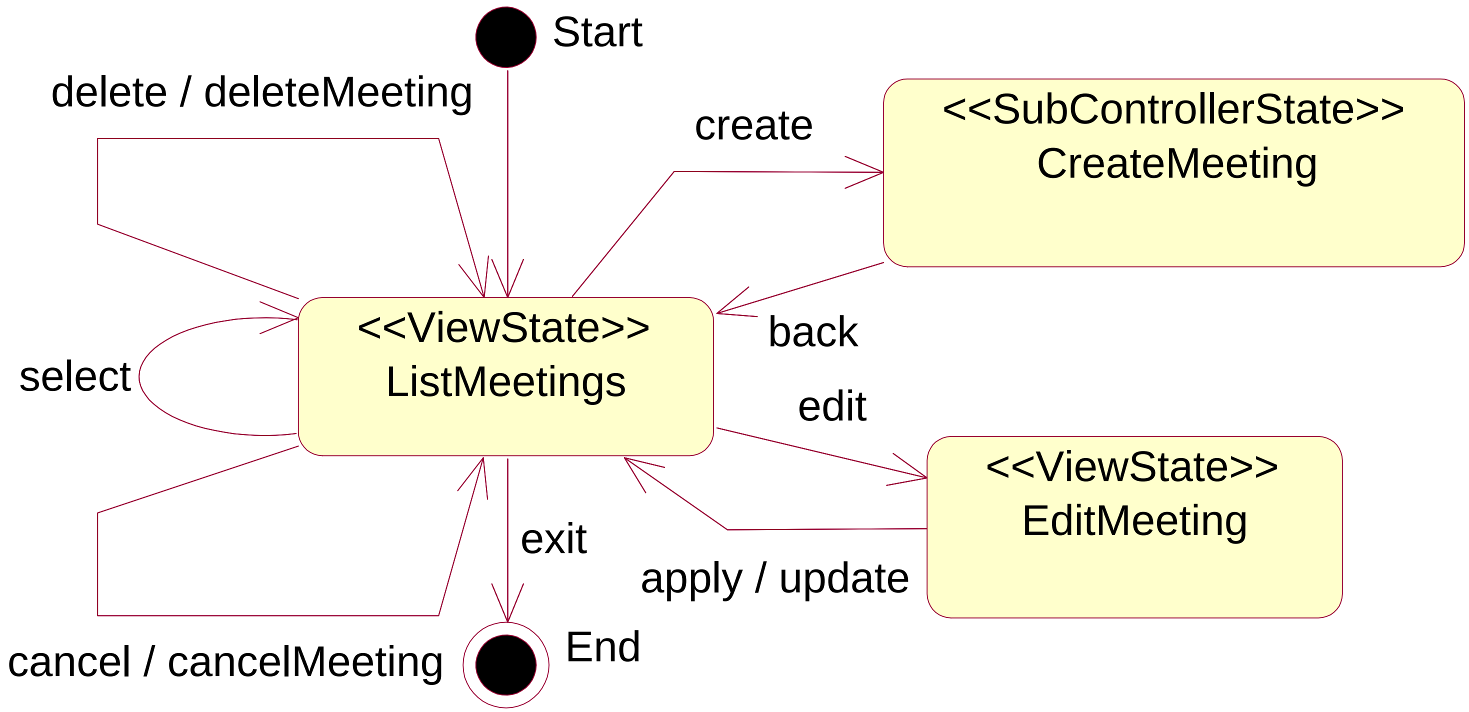


Start: System makes an "epsilon" transition into the state *ListMeetings*.

ListMeetings: User can browse all meeting entries and select one for further processing.

End: The final state of the system after receipt of the exit event.

Statemachine of the MainController



Outgoing transitions (from ListMeetings) include:

- cancel $\mapsto$ cancellation of the selected meeting.
- edit $\mapsto$ transition to EditMeeting, where the selected meeting can be edited.
- create $\mapsto$ transition to CreateMeeting, where a new meeting entry is created (using CreationController).

Secure Controllers

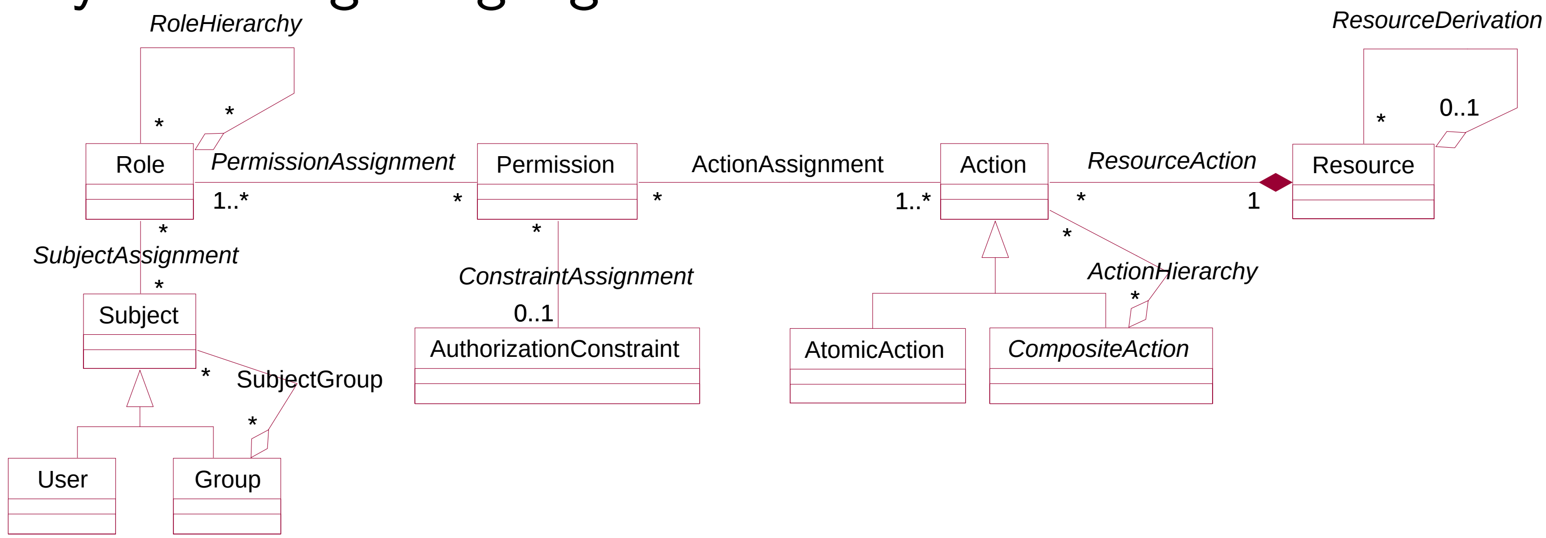
- ControllerUML: a modeling language for controllers.

☞ **Integrating ControllerUML with SecureUML.**

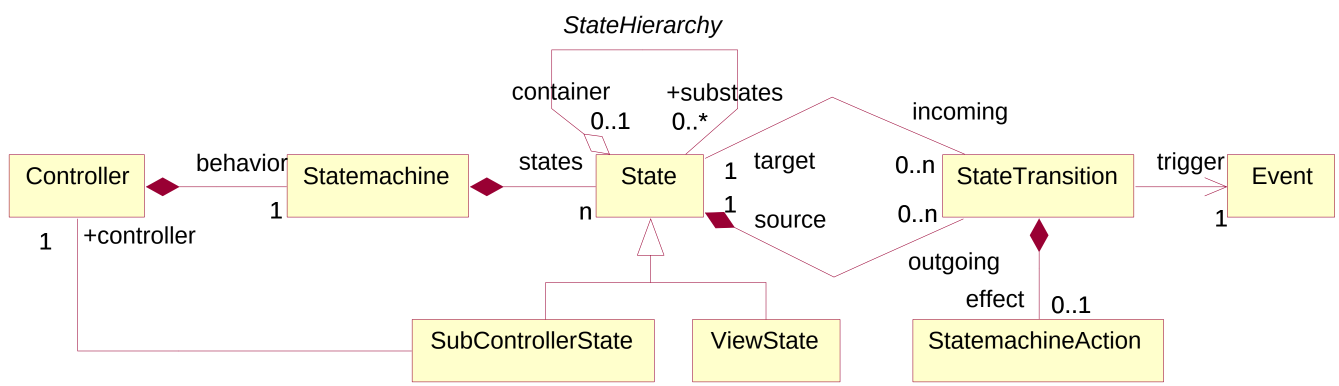
- Generating secure web applications based on the Java Servlet architecture.

Dialect as a Bridge

- Security Modeling Language = SecureUML

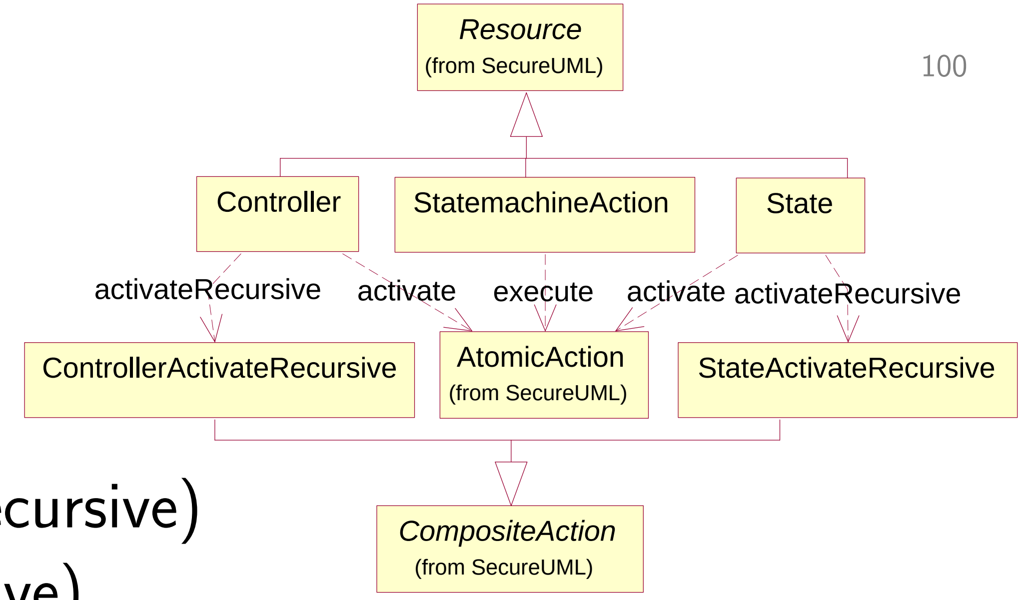


- System Design Modeling Language = ControllerUML



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

Dialect Definition



- Define resources and actions:
  - Controller (activate, activateRecursive)
  - State (activate, activateRecursive)
  - StatemachineAction (execute)
- Define the action hierarchy (in OCL):
  - 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.

LTS Semantics: Idea

- Each controller c in model gives rise to:

  Controller sort C_c, whose elements represent controller instances. (Assumption: finitely many controllers running at any time point).

  State sort S_c, whose elements represent the states of controller's statemachine. and current states.

- Actions are defined by:

  Atomic actions in SecureUML dialect, and

  state-transitions from model.

  These may change controller's data and "current state" attribute

- **That's it!**

  General schema gives semantics for combination with SecureUML

Semantics: Signature $\Sigma_{St} = (\mathcal{S}_{St}, \mathcal{F}_{St}, \mathcal{P}_{St})$

- $\mathcal{S}_{St} = \{C_c \mid c \text{ is a controller}\} \cup \{S_c \mid c \text{ is a controller}\}$
  $\cup \{\texttt{String}, \texttt{Int}, \texttt{Real}, \texttt{Boolean}\}$

- $\mathcal{F}_{St} = \{get_{at} \mid at \text{ is a controller attribute}\} \cup \{\textit{self}_c \mid c \text{ is a controller}\}$
  - Contollers have only attributes at, no methods.
  - Types as expected, e.g., get_{at} has type $s \to v$, where s is controller sort v is sort of attribute's type.
  - Initial and current states of a controller's statemachine are denoted by attributes `initialState` and `currentState` of type S_c.

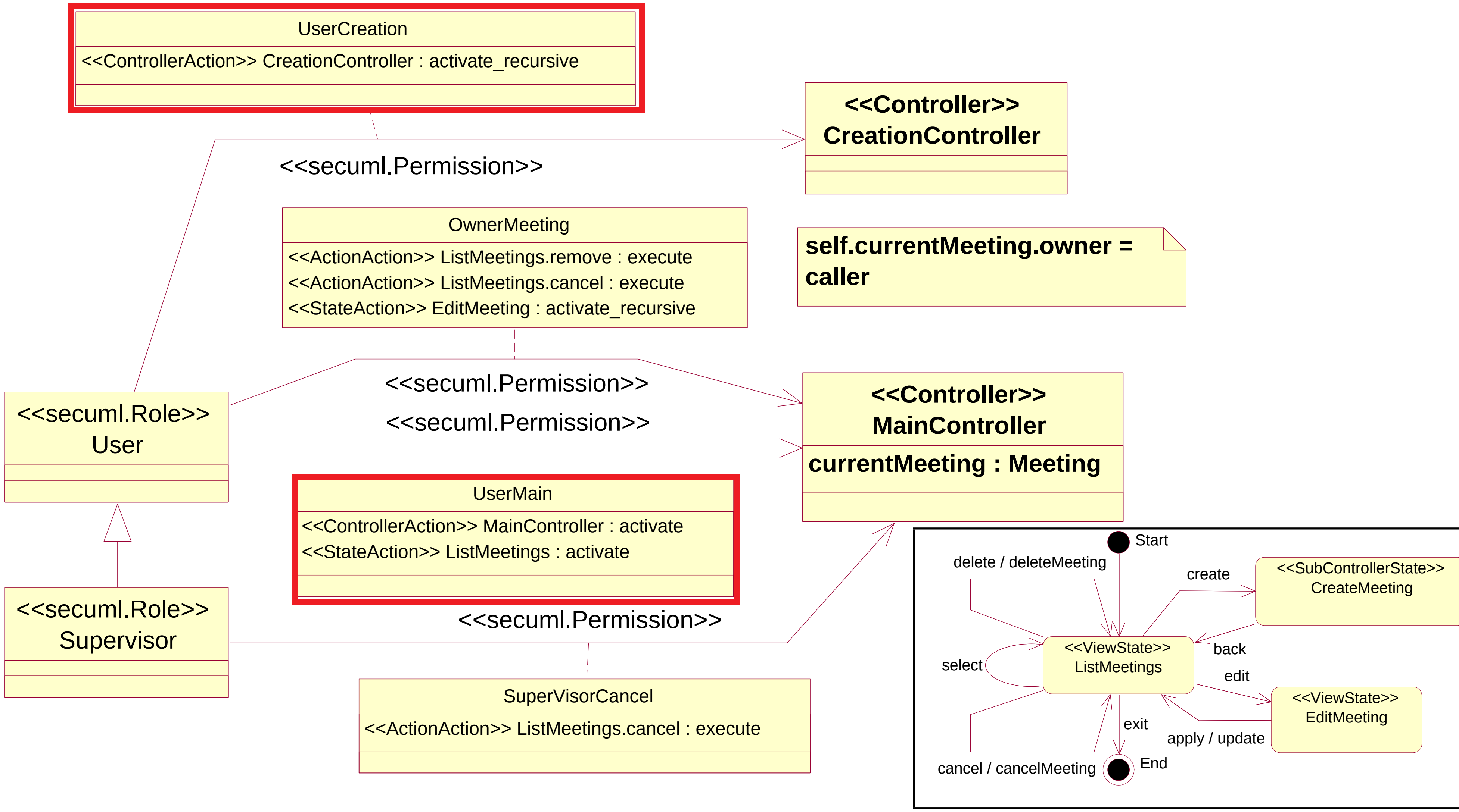
- $\mathcal{P}_{St} = \emptyset$ (no predicate symbols)

Semantics: LTS $\Delta = (Q, A, \delta)$

- Q is set of all first-order structures over the signature Σ_{St} with finitely many elements for each controller sort.
  - Interpretations of `String`, `Int`, `Real`, `Boolean` are standard ones
  - S_c is set of states of controller c
- Actions A correspond to atomic actions defined in dialect (activate controller and state, and execute state machine action) $+$ transitions.
- δ defined via any "standard" state-machine semantics

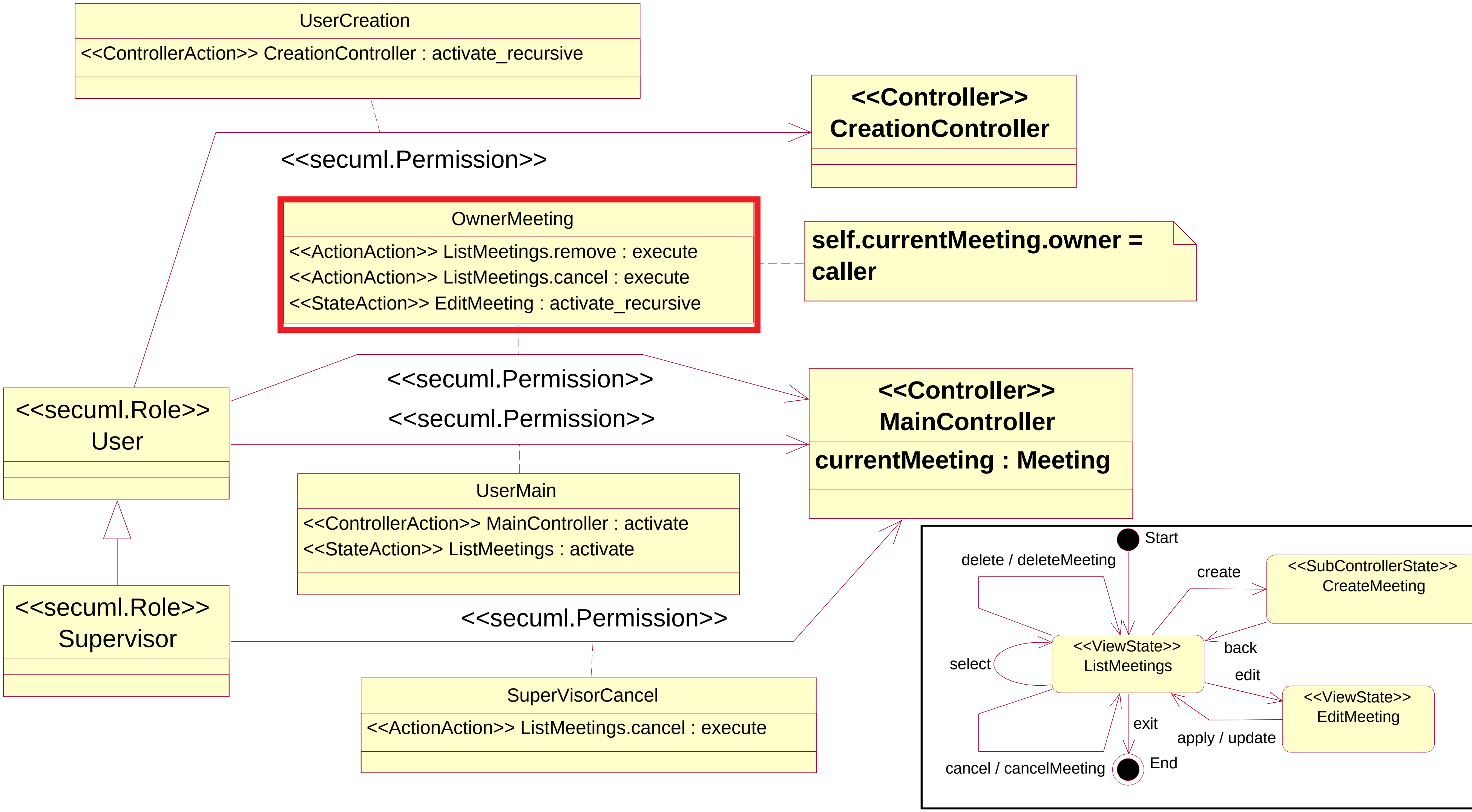
  E.g., for each transition $s_1 \xrightarrow{a} s_2$ in the model there are corresponding tuples (s_{old}, a, s_{new}) in δ, where the current state of the controller is s_1 in s_{old} and is s_2 in s_{new}.
- Can be combined further with ComponentUML by merging the sorts, function and predicate symbols defined by them.

Example Policy: Permissions



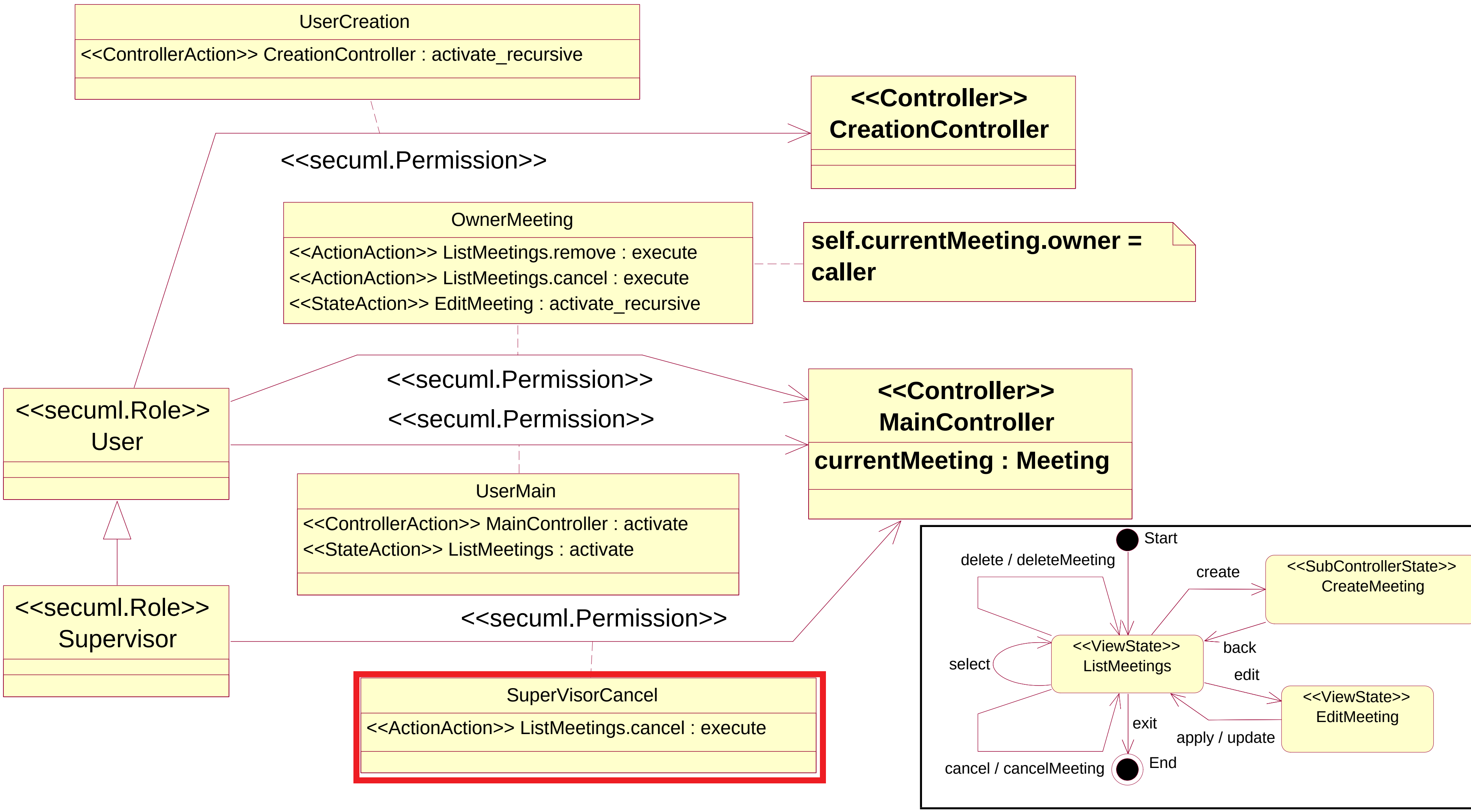
1. All users of the system can create new meetings and read all meeting entries.

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.

Secure Controllers

- ControllerUML: a modeling language for controllers.
- Integrating ControllerUML with SecureUML.

☞ **Generating secure web applications based on the Java Servlet architecture.**

What are Java Servlets?

- A Servlet is (essentially) a Java class that runs on a webserver and is used to implement web-applications, e.g.,
  - process data submitted by HTML forms
  - provide dynamic content (e.g., answers to database queries)
  - manage state information (e.g., your shopping cart)

- Similar to EJBs, Servlets execute in an application server (called "Servlet container", e.g., Tomcat). The Java Servlet specification defines the API for this container.

- Java Servlet provides declarative access control mechanisms (where protected resources are URLs) and programmatic access control mechanisms, based on RBAC.

Overview of Transformation

- **Starting point (MDA):** A transformation function, translating ControllerUML models into web applications based on the Servlet standard.
- **Limitations of the Java Servlet Access Control Architecture:**
  - Declarative Access Control only enforces policies upon requests from the outside, not for requests from Servlets to other Servlets on the same server. This is problematic for some kinds of system architectures, e.g., when using the “front-controller pattern”.
  - As with EJBs, role-inheritance is not supported.
- **Approach:** Basic transformation function is extended by rules, which create an access control infrastructure based on the programmatic access control of Java Servlet.

Basic Transformation Rules

We build on (standard, MDA) transformation rules that transform a ControllerUML model into a DFA, implemented as Java Servlets:

- State $\mapsto$ Singleton class containing information about the state, e.g., enabled transitions. Also includes a method `activate` for activating the state. E.g., for the ListMeetings state:

```
public class ListMeetingsState{
... public void activate(){ ... } ...
}
```

- StatemachineAction $\mapsto$ Java class with a method `perform`, containing the action's "business logic". E.g., for deleteMeeting action:

```
public class deteleMeetingAction{
...  public void perform() { ... } ...
}
```

Basic Transformation Rules (cont.)

- Controller $\mapsto$ Servlet class implementing the control logic as formalized by the controller's statemachine. This includes for example processing events that result in state transitions. Also includes a method `activate` for activating the controller.

```
public class MainControllerServlet extends HttpServlet{
...
    public void activate(){ ... }
...
}
```

Extended Transformation Rules

- For each protected atomic action of ControllerUML there is a corresponding method in the controller implementation.
  - activate of the controller $\mapsto$ `Controller.activate()`
  - activate of the state $\mapsto$ `State.activate()`
  - execute of StatemachineAction $\mapsto$ `StatemachineAction.perform()`
- Add Java assertions to the start of the bodies of these methods, which enforce the policy for the corresponding protected action.
  - necessary authentication information about the current caller are obtained using the programming interfaces for procedural access control of Java Servlet.

How does such an assertions look like?

Generating Assertions

As with componentUML, for each atomic action, compute the set of permissions P for this action and the set of Roles $R(p)$ assigned to each permission $p \in P$:

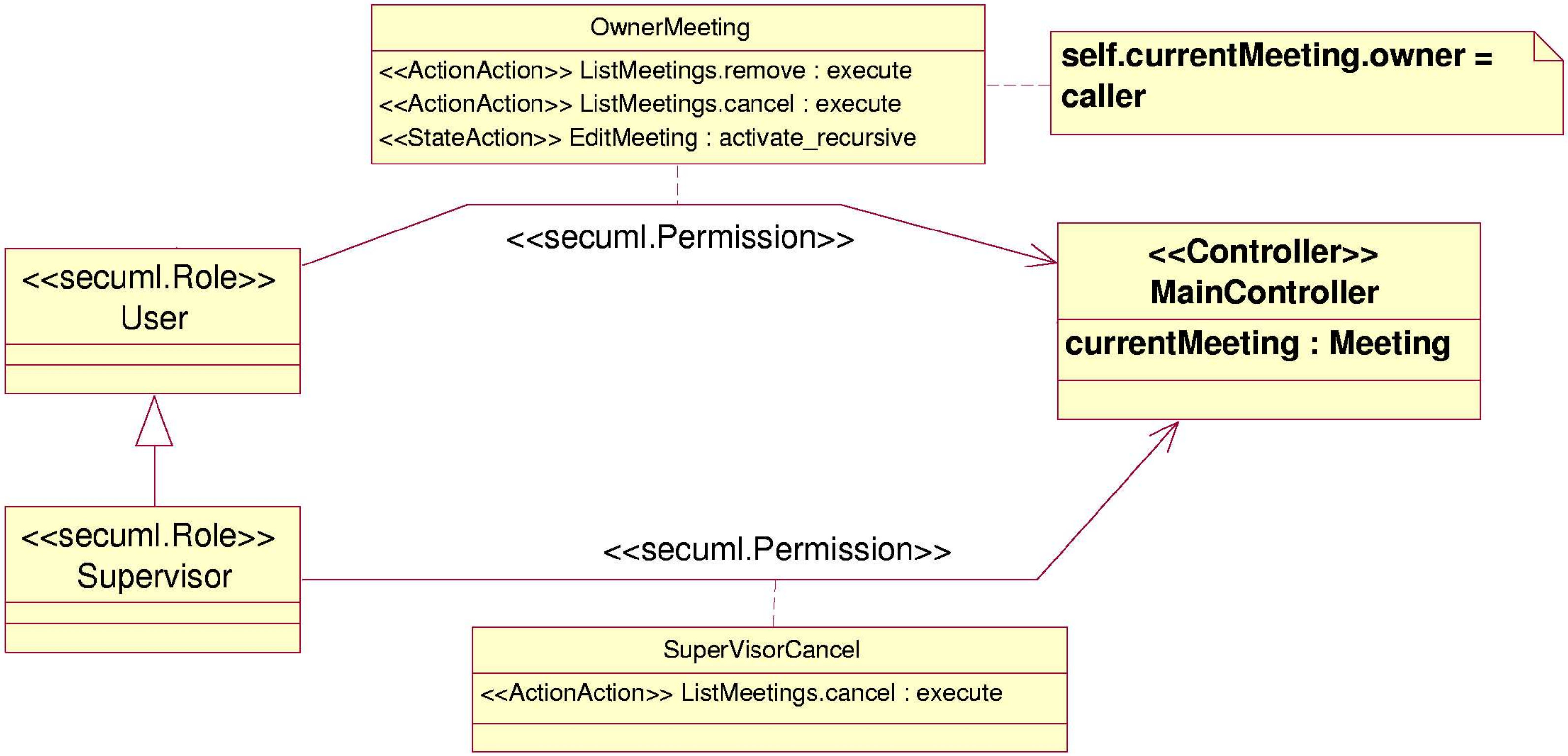
$$P := \{p \in \mathsf{Permissions} \mid (p, a) \in \mathsf{AA} \circ \geq_{\mathsf{Actions}} \circ \mathsf{PA}\}$$
$$R(p) := \{r \in \mathsf{Roles} \mid (r, p) \in \mathsf{PA} \circ \geq_{\mathsf{Roles}}\} \ .$$

Then create an assertion of the form

$$\texttt{if (!(} \bigvee_{p \in P} \Big(\Big(\bigvee_{r \in R(p)} \texttt{request.isUserInRole(}r\texttt{)} \Big) \wedge \mathsf{constraint}(p) \Big) \texttt{))}$$
$$\texttt{c.forward("/unauthorized.jsp");} \ .$$

Denial of access signaled by viewing the error page `unauthorized.jsp`.

Example Assertion



Generated assertion for the action execute on the statemachine action ListMeetings.cancel:

```
if (!(request.isUserInRole("Supervisor") ||
    ((request.isUserInRole("User") || request.isUserInRole("Supervisor")) &&
     getSelectedMeeting().getOwner().getName().equals(request.getRemoteUser()))))
     c.forward("/unauthorized.jsp");
```

Road Map

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

☞ **Experience, demonstration, and conclusions**

Current Status

Foundational:

- Developed idea of Model Driven Security.
- Supports model-centric, generative development.

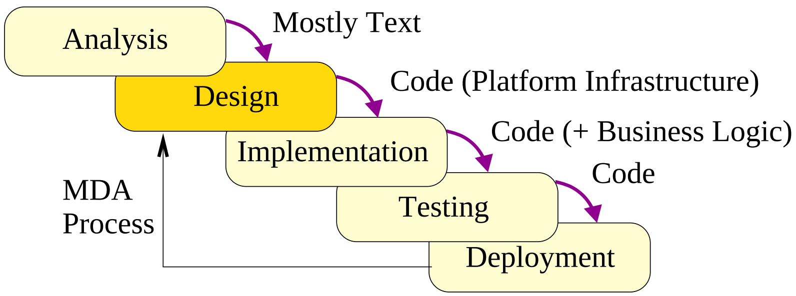
Practical/Tool: Prototype built on top of Rational Rose^{TM}.

- Generators for J2EE (Bea EJB Server) and .NET.
- Industrial version developed by iO GmbH.

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.
- Carried out by Torsten Lodderstedt during his Ph.D.

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 create and read all catalog data, to update their own customer data, to create purchase orders, and to read their own purchase orders.

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, ...

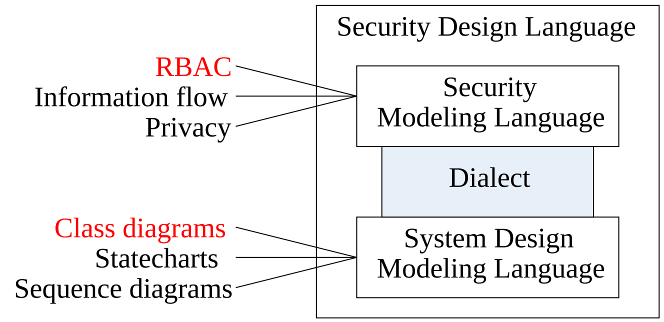
- **Claim:** Least privilege would be not be practically implementable without such an approach.

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

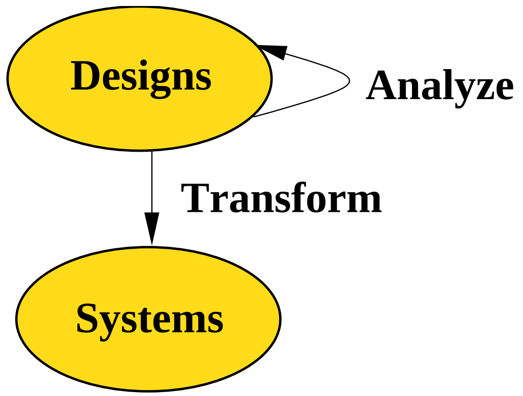
- MDS process provides conceptual support for building models
  - Fits well with a requirements/model-driven development process.
  - Provides a good transition from semi-formal to formal modeling.

Future Work

- Explore the parameter space.
  - ▶ Security/privacy properties.
  - ▶ Modeling languages.



- Exploit well-defined semantics.
  - ▶ Analysis possible at model level.
    Examples: model-consistency, model checking.
  - ▶ So is a verifiable link to code.



⇒ applications to building certifiably secure systems!

Demonstration