The Category-Based Approach to Access Control, Obligations and Privacy

Maribel Fernández
King’s College London

SACMAT 2023

Joint work with
Sandra Alves   Clara Bertolissi   Jenjira Jaimunk   Bhavani Thuraisingham
Motivations

A variety of access control models...

...each with specific languages, techniques, properties.

- RBAC: Role-Based Access Control
- Mandatory (e.g., [Bell-Lapadula] military applications)
- Event-Based (e.g., DEBAC in banking applications)
- ABAC: Attribute-Based Access Control
- ...

⇒ An Access Control MetaModel [Barker09] based on the primitive notion of a category.
Category-Based MetaModel

- Core set of principles of access control, can be specialised for specific applications
- Abstracts away complexities of specific access control models
- Formally defined: axiomatic approach
  - to compare policies rigorously
  - understand the consequences of changes
  - prove properties of policies and combinations of policies
In this talk:

- The category based metamodel
- Category-based
  - Access Control: CBAC
  - Obligations: CBAC-O
  - Privacy: CBDA
- Conclusions and future work
The Category-Based Metamodel: entities, relationships, axioms
Entities

- countable set $\mathcal{C}$ of categories: $c_0, c_1, \ldots$
- countable set $\mathcal{P}$ of principals: $p_1, p_2, \ldots$
- countable set $\mathcal{A}$ of actions: $a_1, a_2, \ldots$
- countable set $\mathcal{R}$ of resources: $r_1, r_2, \ldots$
- countable set $\mathcal{S}$ of situational identifiers (locations, times)

Category: class, group, or domain, to which entities belong

Particular cases:
role, security clearance, attribute-based...
Relationships between entities

- **Principal-category assignment** $\mathcal{PCA}$:
  
  \[(p, c) \in \mathcal{PCA} \text{ iff } p \in \mathcal{P} \text{ is assigned to } c \in \mathcal{C}\]

- **Resource-category assignment** $\mathcal{RCA}$:
  
  \[(r, c) \in \mathcal{RCA} \text{ iff } r \in \mathcal{R} \text{ is assigned to } c \in \mathcal{C}\]

- **Permission-category assignment** $\mathcal{ARCA}$:
  
  \[(a, c_r, c_p) \in \mathcal{ARCA} \text{ iff action } a \in \mathcal{A} \text{ on resource category } c_r \\
  \text{may be performed by principals in the category } c_p\]

- **Authorisations** $\mathcal{PAR}$:
  
  \[(p, a, r) \in \mathcal{PAR} \text{ iff } p \in \mathcal{P} \text{ may perform action } a \in \mathcal{A} \text{ on resource } r \in \mathcal{R}\]
Core axiom:

\[(a1) \quad \forall p \in \mathcal{P}, \forall a \in \mathcal{A}, \forall r \in \mathcal{R},\]

\[
((\exists c_p, c'_p, c_r, c'_r \in C, \\
(p, c_p) \in \mathcal{PCA} \land c_p \subseteq c'_p \land (r, c_r) \in \mathcal{RCA} \land c_r \subseteq c'_r \land \\
(a, c'_r, c'_p) \in \mathcal{ARCA}) \\
\iff (p, a, r) \in \mathcal{PAR})
\]

\(\subseteq\) is a relationship between categories (equality, set inclusion, . . .)

Additional relationships and axioms could be added
CBAC
Category-Based Access Control

A basic category based policy is a tuple $\langle E, PCA, ARCA, PAR \rangle$, where $E = (P, C, A, R, S)$, and axiom (a1) is satisfied.

Expressiveness:
A range of existing access control models can be represented as specialised instances of CBAC [Bertolissi and F 2010]:

Unifying formal model CBAC
general notion of category seen as grouping of entities
Category-Based Access Control

Operational semantics: \((a_1)\) is realised through a set of function definitions (rewrite rules) [Bertolissi and F, 2014]

Why rewriting:

- Expressive, multi-paradigm specification language
- Well-developed theory
- Languages and Tools for rapid prototyping/policy analysis: Maude, Tom, CiME
Rewrite-based specification of the axiom (a1):

\[(a2) \quad \text{par}(P, A, R) \rightarrow \text{if} \quad \text{inARCA}^*(A, \text{contain}(\text{rca}(R)), \text{contain}(\text{pca}(P))) \quad \text{then grant else deny}\]

grant and deny are answers
pca, rca compute the list of categories of a principal/resource
contain computes the set of categories that contain any of the categories in its input
\(\in\) is a membership operator
arca returns the list of all the permissions assigned to the categories in a set
Evaluation

An access request by a principal $p$ to perform the action $a$ on the resource $r$ is evaluated by rewriting $\text{par}(p, a, r)$ to normal form.

Proposition:
$\text{par}(p, a, r) \rightarrow^* \text{grant}$ if and only if $(p, a, r) \in \mathcal{PAR}$
Example policy

Employees in a company are classified as managers, senior managers or senior executives.

To be categorised as a senior executive \((\text{SeniorExec})\), a principal must be a senior manager \((\text{SeniorMng})\) according to the information in site \(\nu_1\) and must be a member of the executive board.

Any senior executive is permitted to read the salary of an employee, provided the employee works in a profitable branch and is categorised as a Manager \((\text{Manager})\).

All managers’ names are recorded locally, and the list of profitable branches is kept up to date at site \(\nu_2\).
Example policy

Specific rules (to add to the generic rules computing par):

\[
pca(P) \rightarrow \text{if SeniorMng} \in pca_{\nu_1}(P) \text{ then (if } P \in \text{ExecBoard } \text{then [SeniorExec] else [SeniorMng]} \text{) else [Manager]}
\]

\[
\text{arca(SeniorExec)} \rightarrow \text{zip-read(managers(profbranch}_{\nu_2})
\]

where

- \text{zip-read}, given a list \( L = [l_1, \ldots, l_n] \), returns a list of pairs \([(\text{read}, l_1), \ldots, (\text{read}, l_n)]\)
- \text{profbranch}, defined at site \( \nu_2 \), returns the list of profitable branches
- \text{manager} returns the name of the manager of a branch \( B \) given as a parameter (managers does the same for a list of branches).
Properties of policies

**Totality**: Each request from a valid principal $p$ to perform a valid action $a$ on a resource $r$ receives an answer.

**Consistency**: For any $p \in \mathcal{P}$, $a \in \mathcal{A}$, $r \in \mathcal{R}$, at most one result is possible for a request $\text{par}(p, a, r)$.

**Soundness and Completeness**: For any $p \in \mathcal{P}$, $a \in \mathcal{A}$, $r \in \mathcal{R}$, an access request by $p$ to perform the action $a$ on $r$ is granted if and only if $p$ belongs to a category that has the permission $(a, r)$.

**Remark**: 
$\text{Totality} + \text{consistency} \equiv \text{confluence, termination, sufficient completeness}$

**Sufficient conditions**: orthogonality [Klop], rpo [Dershowitz], . . .

**Application**: example policy is consistent, total, sound, complete
A policy graph is a tuple \( \mathcal{G} = (\mathcal{V}, E, l_v, l_e) \):

- \( \mathcal{V} \) is a set of nodes;
- \( E \subseteq \{ \{v_1, v_2\} \mid v_1, v_2 \in \mathcal{V} \land v_1 \neq v_2 \} \);
- \( l_v \) is an injective labelling function \( l_v : \mathcal{V} \rightarrow C \cup P \cup A \cup R \);
- \( l_e \) is a labelling function for edges.
A well-typed graph contains only the following kinds of edges:

(a) \( \{v_1, v_2\} \in E \) s. t. \( \text{type}(v_1) = P \land \text{type}(v_2) = C_P \),
    connects principals to categories — edge of type \( PC_P \)

(b) \( \{v_1, v_2\} \in E \) s. t. \( \text{type}(v_1) = C \land \text{type}(v_2) = C \),
    connects categories — edge of type \( CC \)

(c) \( \{v_1, v_2\} \in E \) s. t. \( \text{type}(v_1) = C \land \text{type}(v_2) = A \),
    connects categories to actions — edge of type \( CA \)

(d) \( \{v_1, v_2\} \in E \) s. t. \( \text{type}(v_1) = C \land \text{type}(v_2) = R \),
    connects categories to resources — edge of type \( CRR \)
Relations associated with the graph

\( G \) is a well-typed policy graph

Then:

- \( \mathcal{PCA}_G = \{(lv(v_1), lv(v_2)) \mid \text{type}(\{v_1, v_2\}) = PC_p\} \)

- \( \mathcal{RCA}_G = \{(lv(v_1), lv(v_2)) \mid \text{type}(\{v_1, v_2\}) = RC_R\} \)

- \( \mathcal{ARCA}_G = \{(lv(v_1), lv(v_2), lv(v_3)) \mid \text{type}(\{v_1, v_2\}) = AC_R, \text{type}(\{v_3, v_1\}) = CPA\} \)

- \( \mathcal{PAR}_G \) obtained via path computation (from P to R)

Administrative model: Admin-CBAC in the CBAC metamodel
Implementation

**Figure**: Interface of the prototype: landing page
Implementation

Figure: Test policy
Implementation

Here are some Checks to help Administrators analyse the system state:

- **Are there any left out Resources:**
  No resources are unmatched.

- **Are there any left out Principals:**
  No Principals are unmatched.

- **Are there any left out Actions:**
  No left out actions.

**Figure:** Interface of the prototype: analysis menu
Key findings

Expressive power:

- **entities, relations**: generic approach
- Axiomatisation: takes into account **multi-site systems**, **combination of policies, administration**
- Rewrite-based **operational semantics**: supports formal reasoning/policy analysis
- **Graph-based policy representation**: facilitates implementation/policy visualisation
Obligations
Obligations and Access Control

- Licence agreements
- EU GDPR - Data collection
- US Gramm-Leach-Bliley Act for financial institutions
- Medical policies: access to treatment/consent form
Features of Obligations

- Mandatory action
- Within an interval, defined by temporal constraints or events
- Atomic or compound actions
- May depend on conditions
- Interactions between obligations and permissions: fulfilling the obligations may depend on certain permissions.
- Accountability, if obligations go unfulfilled.
Events - Types, History, Interval

**Event:** an *action* that happened at a specific moment in time.

Event Type/Instance = Generic Event/Event

Examples:

- \(\text{alarmON} = \{\text{act = fire\_alarmON, ward = neurology, happens = 20220621.120000}\}\),
- \(\text{callFireDep} = \{\text{act = call\_FireDEP, ward = neurology, happens = 20220621.120500}\}\),

Generic events include variables:

\(\text{alarmON}[W, T] = \{\text{act = fire\_alarmON, ward = W, happens = T}\}\)

*Event history:* ordered sequence \(h\) of events that happen in a run of the system
Obligations

A *generic obligation* is a tuple \((a, r, ge_1, ge_2, s)\)

- \(a\) action,
- \(r\) object,
- \(ge_1, ge_2\) generic events (interval where the obligation must be fulfilled),
- \(s = (op, sec)\) secondary obligations.

**Example:**

\((alert, firedept, alarmON[W, T], alarmOFF[W, T'], (\land, [o_{call}, o_{notify}]))\),

\(o_{notify} = (notify, headTeam, alarmON[T, W], T + 3, id)\)

\(o_{call} = (call, firedept, alarmON[T, W], alarmOFF[W, T'], id)\)

**Generic vs Concrete obligation**
Duties

A *duty* is a tuple \((p, o)\) of a principal and a concrete obligation.

**Duty status:**

- *invalid*;
- *fulfilled*;
- *violated*;
- *pending*;
Obligations in the CBAC Metammodel

More entities:

- Countable sets $E_v$ and $GE_v$ of specific events and generic events, respectively: $e, e_1, \ldots; ge, ge_1, \ldots$
- Countable set $H$ of event histories, $h, h_1, \ldots$
- Countable sets $O$ of obligations, $o, o_1, \ldots$, and $S$ of subordinate pairs, $s, s_1, \ldots$

The elements of $S$ are either $id$ or pairs of the form $(op, sec)$ where $op$ is an operator and $sec$ is a list of obligations
More relations

Obligation-category assignment:
$\mathcal{OCA} \subseteq \mathcal{O} \times \mathcal{C}$: $(o, c) \in \mathcal{OCA}$ iff $o$ is assigned to principals in $c$.

Obligation-principal assignment:
$\mathcal{OPA} \subseteq \mathcal{O} \times \mathcal{P}$: $(o, p) \in \mathcal{OPA}$ iff $p \in \mathcal{P}$ has the obligation $o$.

Duty assignment:
$\mathcal{DA} \subseteq \mathcal{O}^\varnothing \times \mathcal{P} = \mathcal{D}$, such that $(o, p) \in \mathcal{DA}$ iff the principal $p \in \mathcal{P}$ must fulfil the concrete obligation $o = (a, r, e_1, e_2, s)$.

Examples:
- $o[P, D] = (\text{visa}, \text{passport}(P), \perp, \text{registration}(P, D), \text{id})$
- $\mathcal{OCA}$: $(o[P, D], \text{international-student})$
- $\mathcal{PCA}$: $(\text{JohnSmith}, \text{international-student})$
- $\mathcal{OPA}$: $(o[\text{JohnSmith}, D], \text{JohnSmith})$
- $\mathcal{DA}$: $(o[\text{JohnSmith}, 20.09.2022], \text{JohnSmith})$
The relations \textit{FULFILLED}, \textit{PENDING} and \textit{VIOLATED} are also axiomatised.
Analysis of policies

- models of access control and obligation
- authorisations and obligations co-exist: interdependencies
- dynamic categories: e.g. depending on events in $h$

Checking interactions: every user has the required permissions in order to fulfill the duties

**Strong and Weak Compatibility:** Sufficient conditions to ensure that only duties that are consistent with authorisations are issued.
Privacy
large quantities of data are transmitted to external services

Benefits:
+++ collected data can be used to provide better services to users

Risks:
--- security and privacy

Goal: **Users control which/when data is collected and shared**
   Supported by regulations: GDPR in EU, FTC in US, etc.
   Techniques??
   encryption/differential privacy...useful but not sufficient
Controlling Data-Collection and Data-Sharing

Two key notions:
- control the way data is collected/transmitted
- control access to data

Requirements:
a cloud-IoT architecture with

integrated data-collection, storage and data management model +
policy languages, enforcement mechanisms, reasoning techniques

Challenges:
variety of IoT devices and services, variety of users, policy
specification and enforcement
DataBank: A Privacy-Preserving Cloud-IoT Architecture

Main features:
- **data repositories**: cloud + local Data Pocket
- **data collection control** before uploading to the cloud
- **access control** to restrict access to data by third parties
- Implemented by Privasee
Category-Based Data Access Model (CBDA)

Entities:

- \( \mathcal{D} \): data sources
- \( \mathcal{DI} \): data items e.g. location, time, speed
- \( \mathcal{S} \): external services that process data
- \( \mathcal{C} \): categories partitioned into
  - \( \mathcal{C}_{DU} \): unprocessed data
  - \( \mathcal{C}_{DS} \): stored data for sharing
  - \( \mathcal{C}_S \): services
- \( \mathcal{A} \): actions, partitioned into
  - \( \mathcal{A}_D \): data collection actions e.g., upload, average, encrypt
  - \( \mathcal{A}_S \): service actions on stored data, e.g., view, transfer

Categories can be dynamic: defined via attributes
CBDA Model

Relationships:

- **Device-Data Assignment**: \( DUA \subseteq D \times DU \)
- **Data Item-Category Assignment**: \( DICA \subseteq DI \times C \), partitioned into \( DICA_U \) and \( DICA_S \)
- **Action-Category Assignment**: \( ACA \subseteq A \times C \times C \), partitioned \( ACA_D \) (data collection actions) and \( ACA_S \) (service actions):
- **Service-Category Assignment**: \( SCA \subseteq S \times C \)
- **Authorised Data Collection**: \( AD \subseteq A \times DU \times DS \) \((da, ud, sd) \in AD \) iff the data collection action \( da \in AD \) is authorised on \( ud \in DU \) to produce \( sd \in DS \).
- **Authorised Data Access**: \( ADS \subseteq A \times DS \times S \), such that \((sa, sd, s) \in ADS \) iff service action \( sa \in AS \) is authorised on the stored data item \( sd \in DS \) for the service \( s \in S \).
CBDA Model

Axioms for authorisations (simplified: no category hierarchies)

\[
\text{Data Collection: unprocessed data } \rightarrow \text{ stored data}
\]

\( (da1) \quad \forall ud \in \mathcal{D}_U, \forall sd \in \mathcal{D}_S, \forall da \in \mathcal{A}_D, \)
\[\exists udc, dsc \in \mathcal{C}, (ud, udc) \in \mathcal{DICA}_U \land \]
\[ (da, udc, dsc) \in \mathcal{ACA}_D \land (da, ud, sd) \in \mathcal{OP} \land \]
\[ (sd, dsc) \in \mathcal{DICA}_S ) \iff (da, ud, sd) \in \mathcal{AD} \]

\[
\text{Data Access: stored data } \rightarrow \text{ services}
\]

\( (da5) \quad \forall sd \in \mathcal{D}_S, \forall sa \in \mathcal{A}_S, \forall s \in S, \)
\[\exists dsc, sc \in \mathcal{C}, (sd, dsc) \in \mathcal{DICA}_S \land (s, sc) \in \mathcal{SCA} \land \]
\[ (sa, dsc, sc) \in \mathcal{ACA}_S ) \iff (sa, sd, s) \in \mathcal{ADS} \]
Graph-Based CBDA Policies

CBDA *policy graphs*:
- nodes represent policy *entities*,
- edges represent *relations* defining how entities are categorised and authorised/prohibited actions for each category of entities.

Graph elements are labelled

Types of nodes in a CBDA graph:
Graph-Based CBDA Policies

Well-typed graphs represent policies. Paths of specific types define the authorised and prohibited actions for each kind of data item and service.

Authorisation Path:

Prohibition Path:
Example CBDA Policy Graph
CBDA Policy Analysis/Queries and Enforcement

Policy queries answered by graph traversal.
Example Policy Content Query:
Are there (permitted or banned) actions assigned to each category? 
Answer:
All the categories have some associated action if and only if each node $v$ of type $C$ is in an authorisation or prohibition path.

Example Policy Effect Queries: Absence of conflict (mutually exclusive actions $a_1$, $a_2$ on $d_i$ are not permitted).
Answer:
Set of authorisation paths starting in $d_i$ does not contain paths via $a_1$ and paths via $a_2$.

Enforcement of privacy preferences:
Service obligation policy matches CBDA policy
Conclusions - Future work

• Categorisation: powerful abstraction mechanism
• Future work:
  • Policy languages / Usability
  • Policy enforcement / obligations: privacy
  • Negotiation: Risk-Benefit Analysis - optimal policy
  • Policy Mining
  • Policy composition . . .
Conclusions - Future work

- **Categorisation:** powerful abstraction mechanism
- **Future work:**
  - Policy languages / Usability
  - Policy enforcement / obligations: privacy
  - Negotiation: Risk-Benefit Analysis - optimal policy
  - Policy Mining
  - Policy composition . . .

This talk is based on:
- Bertolissi and F. *A metamodel of access control for distributed environments.* Information and Computation 2014
- Alves and F. *A graph-based framework for the analysis of access control policies.* Theoretical Computer Science 2017
- Alves and F. *An Expressive Model for the Specification and Analysis of Obligations.* Preprint 2023