UNIVERSITY
AT ALBANY
State University of New York

VisualOFGA: Visualizing Open Fine Grained Authorization Policies

Thursday 01th May, 2025

Introduction to ReBAC

ReBAC is an access control model that defines permissions based on the relationships between entities and subjects of your system.

Components of Rebac

Relevant Resources

1. **Entities:** These are the core objects within your system, such as users, projects, teams, or resources
2. **Relations:** These define the connections or relationships between entities, for example, a user being the ”creator” of a project or a team having ”access” to a resource.
3. **Policies:** ReBAC policies are defined around the relationships between entities, allowing access based on these relationships, such as a user owning a document, or a team having access to a project.

Introduction to OpenFGA

OpenFGA is an open-source authorization engine that implements ReBAC at scale. Inspired by Google Zanzibar, it allows developers to define and evaluate relationship-based policies.

Core Features

1. Declarative policy definition using DSL or JSON
2. Entity types, relations, and permissions
3. Logical composition: `or`, `and`
4. Delegation via the `from` keyword
5. Tuple-based access evaluation

Example

“User A can write to Repo R if they are a member of a group that owns R.”
OpenFGA resolves this via relationship graph traversal at runtime.

Problem Statement i

Problem 1: Schema Complexity

Issue: OpenFGA schemas written in DSL/JSON become hard to interpret, especially with nested relationships and delegation.

Contribution: Visual Schema Rendering - we convert the schema into interactive graphs to make relationships and logic more transparent.

Problem Statement ii

Problem 2: Limited Visualization

Issue: OpenFGA currently offers only static type previews or query-based path graphs, which don't help understand full schema logic.

Contribution: Full Relationship Flow Visualization — we visualize all entity types, relations, inheritance, and delegation paths.

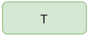
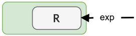
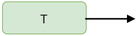
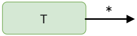
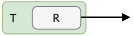
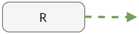
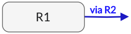
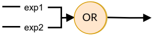
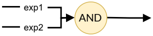
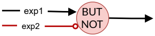
Problem Statement iii

Problem 3: No Visual Editing

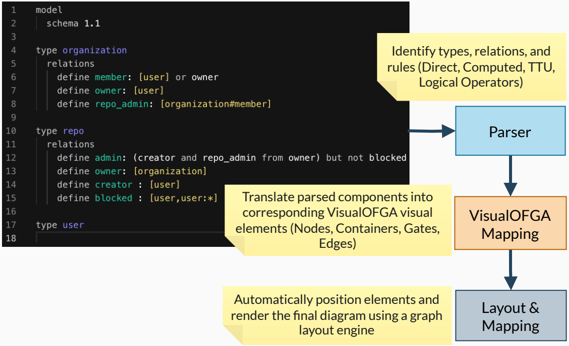
Issue: There's no way to modify policies visually — editing must be done manually in DSL or JSON.

Contribution: Graph-to-Schema Reversibility — users can modify the visual graph and convert it back into valid schema definitions.

Visual Notation Summary

Category	OpenFGA Construct / Terminology	Formal SDL Syntax	Visualization
Definition	Type Definition	type T	
	Relation Definition	define R: exp	
Base Expression	Direct Relationship Type Restrictions (Direct Type Assignment)	[T]	
	Direct Relationship Type Restrictions (Type Bound Public Access)	[T:*]	
	Direct Relationship Type Restrictions (Direct Userset Assignment)	[T#R]	
	Referencing Other Relations On The Same Object (Computed Userset)	R	
	Referencing Relations On Related Objects (TupleToUserset)	R1 from R2	
Logical Operator	Union Operator (or)	exp1 or exp2	
	Intersection Operator (and)	exp1 and exp2	
	Exclusion Operator (but not)	exp1 but not exp2	

Methodology i



```
model
  schema 1.1

type organization
  relations
    define member: [user] or owner
    define owner: [user]
    define repo_admin: [organization#member]

type repo
  relations
    define admin: (creator and repo_admin from owner) but not blocked
    define owner: [organization]
    define creator : [user]
    define blocked : [user,user:*]

type user
```

Parser

Identify types, relations, and rules (Direct, Computed, TTU, Logical Operators)

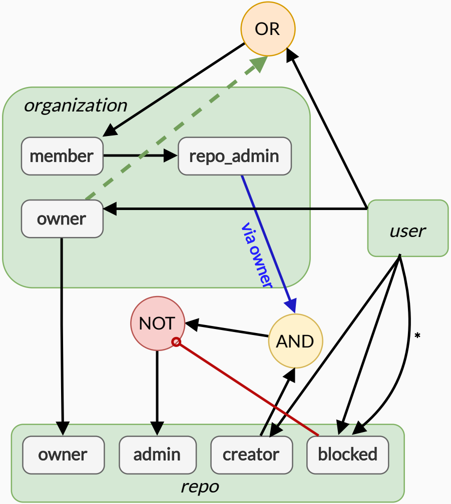
VisualOFGA Mapping

Translate parsed components into corresponding VisualOFGA visual elements (Nodes, Containers, Gates, Edges)

Layout & Mapping

Automatically position elements and render the final diagram using a graph layout engine

Methodology ii



Progress and Future Plan i

Current Progress

We have successfully developed the first version of our project.

1. Implemented OpenFGA Playground parser.
2. Developed graph visualizations using nodes and edges, with distinct edge styles representing different types of relationships between nodes.
3. Enabled copying of JSON and DSL files.

These implementations effectively address Problem Statements I and II.

Progress and Future Plan ii

Future Plan

1. Implement dynamic alignment for nodes and edges to improve visual clarity
2. We aim to enable visual editing, allowing users to interact with and modify the graph directly, which will successfully address Problem Statement III.

THANK YOU :)