How are relationships validated during registration (model import)

Relationships in Meshery

Meshery Relationships characterize how components are connected and interact with each other. Relationships are defined within models to aid in structuring the interrelationships between one or more components in a design to further aid in comprehending the overall structure and dependencies within managed systems. A relationship might declare that a Namespace parents a Pod, that a RoleBinding binds a Role to a ServiceAccount, or that a PersistentVolumeClaim is mounted into a Deployment.

Every relationship is a JSON or YAML object conforming to relationships.meshery.io/v1aplha3. Before we move forward with the document, it is essential for us to understand the structure of the relationships.

```
// Source: meshery-schemas/models/v1alpha3/relationship/relationship.go
type RelationshipDefinition struct {
    Id              uuid.UUID
    Kind            RelationshipDefinitionKind       // "edge" or "hierarchical"
    Type            string                           // "binding", "non-binding", "parent", "sibling"
    SubType         string                           // "firewall", "inventory", "alias", etc.
    SchemaVersion   string                           // "relationships.meshery.io/v1alpha3"
    Version         string                           // e.g., "v1.0.0"
    Status          *RelationshipDefinitionStatus    // "enabled", "identified", "approved", etc.
    Model           model.ModelReference             // which model this relationship belongs to
    Selectors       *SelectorSet                     // defines how components are matched
    Metadata        *RelationshipDefinitionMetadata  // description, styles, isAnnotation
    EvaluationQuery  *string                         // custom Rego query (usually empty)
}
```

Import Pathways for models

There are four ways a model (and its relationships) can enter Meshery. All four converge at the same registration point.

1) CSV Import
2) File Upload Import
3) URL Import
4) OCI Artifact Import

All four pathways converge at `registrationHelper.Register(dir).` This function walks a standardized directory structure.

```
modelName/
└── v1.0.0/
    └── 1.0.0/
        ├── model.json
        ├── components/
        │   ├── deployment.json
        │   └── service.json
        └── relationships/
            ├── binding-permission.json
            └── parent-inventory.json
```

Relationship Validation During Registration

Currently, the import-time validation is lightweight and the real evaluation happens during the design evaluation, where the OPA/Rego policy engine takes over. The process happening when user imports a model using `mesheryctl model import` can be categorized into:

1) JSON Schema Validation:
   This is the first and primary validation gate. The raw bytes from the relationship definition file are deserialized into the Go struct. It checks:
   1) Whether JSON is syntactically valid
   2) Field types are compatible (strings where strings are expected, arrays where arrays are expected)

   If deserialization fails, the handler returns StatusBadRequest. If it succeeds we move on.
   Source: server/handlers/relationship_handlers.go:185

2) Database Insert
   This is a raw database INSERT. There is no model existence check at this level. The isModel error flag name is misleading, it fires when en.Create() fails, which for relationships would only happen due to a database-level constraint violation (e.g., a foreign key error), not because the system looked up whether a model named "kubernetes" exists in the registry. If you register a relationship with “model” : { “name” : “kibernetes”}, it will still pass.

3) OPA Policy Reload:
   After a successful registration, the system rebuilds the entire OPA Rego policy instance. This loads all currently registered relationships into the Rego engine as data.relationships, making them immediately available for design evaluation.

```
policies.SyncRelationship.Lock()
rego := policies.Rego{}
r, err := policies.NewRegoInstance(models.PoliciesPath, h.registryManager)
if err != nil {
    h.log.Warn(ErrCreatingOPAInstance(err))
} else {
    rego = *r
}
h.Rego = &rego
policies.SyncRelationship.Unlock()
```

Source:https://github.com/meshery/meshery/blob/5c9239243d1e6a465c5d9b5614ad60f7ddbd73d2/server/handlers/component_generator_helper.go#L257-L268

Limitations

The evaluation during registration is minimal and the real heavy-lifting happens when a design is created and the OPA engine takes over.

The import command performs exactly one semantic check: the schemaVersion field must be present and match a recognized prefix. Beyond this, validation is limited to what json.Unmarshal enforces, valid JSON with compatible field types. Everything else passes through unchecked. Some of which are listed below:

1) Kind, Type, and SubType are not enumerated. These fields accept arbitrary strings.
2) Selectors accept null. A relationship with selectors: null is written to the registry without error. Currently being fixed (https://github.com/meshery/meshery/pull/17724)