Query-based Access Control for Secure Collaborative Modeling using Bidirectional Transformations

Csaba Debreceni, Gábor Bergmann, István Ráth, and Dániel Varró

Abstract—Large-scale model-driven system engineering projects are carried out collaboratively. Engineering artifacts stored in model repositories are developed in either offline (checkout-modify-commit) or online (GoogleDoc-style) scenarios. Complex systems frequently integrate models and components developed by different teams, vendors and suppliers. Thus confidentiality and integrity of design artifacts need to be protected by access control policies.

We propose a rule-based technique for secure collaborative modeling where fine-grained access control for models can be defined by model queries, and such access control policies are strictly enforced by bidirectional model transformations. Each collaborator obtains a filtered local copy of the model containing only those model elements which they are allowed to read; write access control policies are checked on the server upon submitting model changes. Rule-based policies are inherently subject to conflicts between the access control rules; these conflicts should be interpreted in a consistent but also predictable way that caters to the preferences of the policy engineer. We propose a deterministic, parameterizable resolution strategy between conflicting rules to calculate effective access permissions for each model fact in the model.

Our approach is illustrated using multiple case studies of the MONDO EU project on which scalability assessments are also carried out. Finally, we introduce the tool support of online collaboration with on-the-fly change propagation and the integration of our approach into existing version control systems to support offline collaboration.

Index Terms—Collaborative Modeling, Access Control, Online, Offline

1 Introduction

The adoption of model driven engineering (MDE) by system integrators (like airframers or car manufacturers) has been steadily increasing in the recent years [1], since it enables to detect design flaws early and generate various artifacts (source code, documentation, configuration tables, etc.) automatically from high-quality system models.

The use of models also intensifies collaboration between distributed teams of different stakeholders (system integrators, software engineers of component providers/suppliers, hardware engineers, certification authorities, etc.) via model repositories, which significantly enhances productivity and reduces time to market. An emerging industrial practice of system integrators is to outsource the development of various design artifacts to subcontractors in an architecture-driven supply chain.

Collaboration scenarios include traditional *offline collaborations* with asynchronous long transactions (i.e. to check out an artifact from a version control system and commit local changes afterwards) as well as *online collaborations* with short and synchronous transactions (e.g. when a group of collaborators simultaneously edit a model, similarly to well-known on-line document / spreadsheet editors). Several collaborative modeling frameworks (like CDO [2], EMFStore [3], etc.) exist to support such scenarios.

However, such collaborative scenarios introduce significant challenges for security management in order to protect the Intellectual Property Rights (IPR) of different parties. For instance, the detailed internal design of a specific component needs to be hidden to competitors who might supply a different component in the overall system, but needs to be revealed to certification authorities in order to obtain airworthiness. Large research projects in the avionics domain (like CESAR [4] or SAVI [5]) address certain collaborative aspects of the design process (e.g. by assuming multiple subcontractors), but security aspects are restricted to that of the system under design. However, an increased level of collaboration in a model-driven development process introduces additional confidentiality challenges to sufficiently protect the IPR of the collaborating parties, which are either overlooked or significantly underestimated by existing initiatives.

Even within a single company, there are often teams with differentiated responsibilities, areas of competence and clearances. Such processes likewise demand confidentiality and integrity of certain modeling artifacts.

Existing practices for managing access control of models rely primarily upon the access control features of the backend repository. *Coarse-grained access control policies* aim to restrict access to the files that store models. For instance, EMF models can be persisted as standard XMI documents, which can be stored in repositories providing file-based access and change management (as in SVN [6], CVS [7]). *Fine-grained access control policies* may restrict access to the model on the row level (as in relational databases) or triple level (as in RDF repositories).

Unfortunately, security policies are often captured directly on the storage (file) level instead of the metamodel and/or the model level, which result in inflexible fragmentation of models in collaborative scenarios. To illustrate the problem (Fig. 1), let us consider two collaborators, a *SW Provider*$_1$ and *HW Supplier*$_1$, each of which has full control over a model (fragment). Now if the *HW Supplier*$_1$ would

• *This paper is partially supported by the EU Commission with project MONDO (FP7-ICT-2013-10), no. 611125. and the MTA-BME Lendület 2015 Research Group on Cyber-Physical Systems.*

like to share part of its model with the $SW\ Provider_1$, then he needs to give access to the entire model or split his model into two, and give access to only one fragment. In case of multiple $SW\ Provider_i$, the same argument applies, which results in a fragmented model for the $HW\ Supplier_1$. In industrial practice, AUTOSAR models may be split into more than 1000 fragments, which poses a significant challenge for tool developers. A significant cause of this fragmentation has roots in confidentiality and access control.

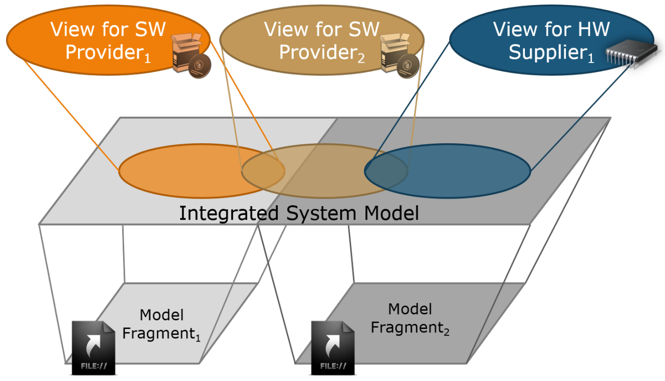


Figure 1: Problem with File-level Access Control

Furthermore, some model persistence technologies (such as EMF's default XMI serialization) do not allow model fragments to cyclically refer to each other, putting a stricter limit to fragmentation, while certain MDE use cases often demand the ability to give access to each object (or even each property of each object) independently.

In addition, these solutions lack flexibility, especially, when accessing models from heterogeneous information sources in different collaboration scenarios. For instance, coarse-grained access control disallows type-specific access control, i.e., to grant or restrict access to model elements of a specific type (e.g., to all classes in a UML model), which are stored in multiple files.

On the other hand, fine-grained access control easily allows to set up inconsistent or insecure access policies. With fine-grained policies, it may be difficult to express how permissions granted for a single model object depend on the context of the object; simply considering the type of model artifacts may not be sufficient.

1.1 Goals and Contributions

The main objective of the paper is to propose fine-grained access control techniques for secure collaborative modeling by advanced model query and transformation techniques while relying upon existing storage back-ends to follow current industrial best practices. In particular, we aim to address the following high-level goals (refined later in Sec. 2):

G1 *Attribute and Relation Based Access Control Policy* that can grant or deny read and write permissions to users for each model object, attribute or cross-reference separately, and where access decisions may depend on property values of model elements, or even their wider context.

G2 *Secure and Versatile Offline Collaboration* where each collaborator can work with a standard representation of the model fragment visible to only them, and process it using standard MDE tools (e.g. editor, verifier). A user may be disconnected from any server or access control mechanism, and then submit his updated version in the end.

G3 *Secure and Efficient Online Collaboration* where multiple users can collaboratively view and edit a model hosted on a server repository in real-time while imposing different read and write permissions. small changes performed by one collaborator are quickly and efficiently propagated to the views visible to other users, without reinterpreting the entire model.

In this paper, we propose a query-based approach for modeling fine-grained access control policies, and define *bidirectional model transformations* to derive *filtered views* for each collaborator and to propagate changes introduced into these views back to a server. The transformation uniformly enforces *high-level fine-grained access control policies* during the derivation of views and the *back-propagation* of changes. It can be used in either *live (incremental)* or *batch* mode to support online and offline collaborative scenarios, respectively. An initial scalability evaluation is carried out using models from the Wind Turbine Case Study of the MONDO European FP7 project, which acts as a motivating example.

Major contributions of this paper with respect to our previous work [8], [9] are (i) the description of the policy language; (ii) the presentation of tool support for secured off-line and on-line collaboration and (iii) a detailed scalability evaluation of our approach in off-line and on-line scenarios.

1.2 Structure

Rest of the paper is organized as follows. Our motivating example is detailed and the challenges are introduced in in Sec. 2. Sec. 3 defines how models can be decomposed into individual assests, and provides a policy language to specify access control rules for assets. In Sec. 4, we overview our bidirectional model transformation for access control. In Sec. 5, we give a brief overview of tool support for collaborative modeling in online and offline scenarios. Sec. 6 describes the evaluation of our approach while related work is overviewed in Sec. 7. Finally, Sec. 8 concludes our paper.

2 Case Study

2.1 Modeling Language

Our approach will be illustrated using a simplified version of a modeling language for system integrators of offshore wind turbine controllers, which is one of the case studies of the MONDO EU FP7 project [10]. The metamodel, defined in Ecore [11] and depicted by Fig. 2, describes how the system is modeled as modules providing and consuming signals that send messages after a specific amount of time defined by the frequency attribute. Modules are organized in a containment hierarchy of composite modules shipped by external vendors, and ultimately containing control unit modules responsible for a given type of physical device (such as pumps, heaters or fans) with specific cycle priorities. For external engineers, a documentation is attached to each signal to clarify its responsibilities. Some of the signals are treated as confidential intellectual property.

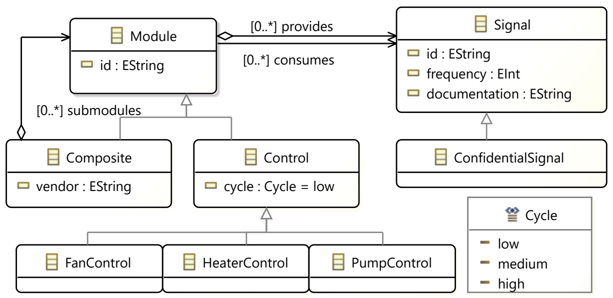


Figure 2: Simplified Metamodel of Wind Turbine Controllers

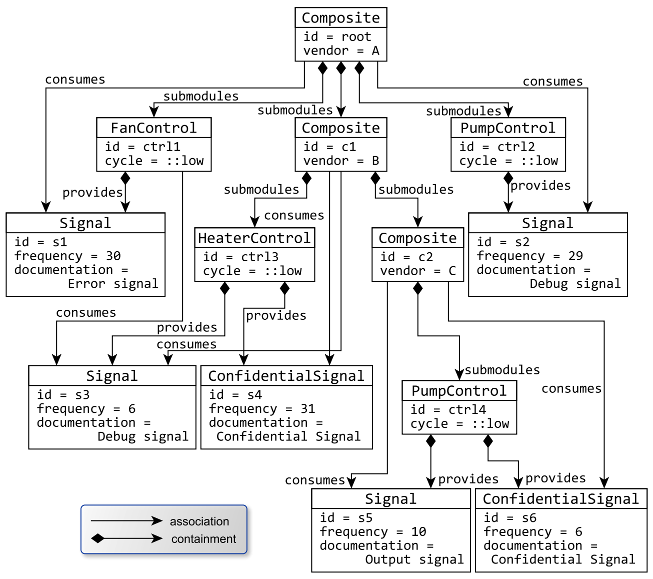


Figure 3: Sample Wind Turbine Instance Model

A sample instance model containing a hierarchy of 3 `Composite` modules with 4 `Control` units as submodules, providing 6 `Signal`s altogether where two of them are `Confidential Signal`s, is shown on Fig. 3. Boxes represent objects (with attribute values as entries within the box). Arrows with diamonds represent containment edges, while arrows without diamonds represent cross-references.

2.2 Security Requirements

Designing wind turbine control units requires specialized knowledge. There are 3 kinds of control units, and each kind can only be modified by specialist users with the appropriate qualification: *heater*, *pump* and *fan control engineers*. Specialist are not allowed to modify (and in some cases, read) parts of the model that would require a different kind of qualification. For this purpose, the following security requirements are stated in relation with control unit specialists:

R1 Each group of specialists shall be responsible for a specific kind of control unit (*owned control units*).

R2 Specialists must see only those signals that are in scope for their owned control units, i.e. signals provided by a module that is either (a) a composite that directly contains an owned control unit, or (b) any submodule (incl. the owned control unit) contained *transitively* in such a composite.

R3 Specialists must be able to modify signals provided by their owned control units.

R4 Specialists must see which modules consume signals provided by their owned control units.

R5 Specialists must not able to see confidential signals.

2.3 Usage Scenarios

The system integrator company is hosting the wind turbine control model on their collaboration server, where it is stored, versioned, etc. There are two ways for users to interact with it.

A group of users may participate in **online collaboration**, where they are continuously connected to the central repository via an appropriate client (e.g. web browser). Through their client, each user sees a live view of those parts of the model that they are allowed to access. In case of the Heater Control Engineer, this view will consist of the elements depicted on Fig. 5a with the following notations: Objects with blue labels and bold borders are visible and modifiable, orange labels and simple borders means visible but not modifiable objects, but elements with dashed borders are hidden from the secured view. Bold edges are visible and writable references, while simple edges are visible but not writable references and dashed edges are hidden from the secured view. In case of attributes, they can be obfuscated or hidden from the view, where the former kind marked with an "O" letter in a square while the latter one marked with an "H" letter in a square. Moreover, Fig. 5b depicts the view associated to the Pump Control Engineer, while Fig. 5c shows the secured view of the Fan Control Engineer.

The users can modify the model through their client, which will directly forward the change to the collaboration server. The server will decide whether the change is permitted under write access restrictions. If it is allowed, then the views of all connected users will be updated transparently and immediately, though the change may be filtered for them according to their read privileges. For example, when the Pump Control Engineer uses their web browser to add a new signal provided by pump control `ctrl2`, then the Heater Control Engineer will not see this change, but if the engineer adds a new signal provided by pump control `ctrl4`, views of the Heat Control Engineer will immediately show the new element.

Alternatively, a user may choose **offline collaboration**. When connecting to the server, they can download a model file containing those model elements that they are allowed to see. The user can then view, process, and modify their downloaded model file locally, with software that can be an unmodified off-the-shelf tool, and need not be aware of collaboration and access control. For example, the Heater Control Engineer can change the `cycle` attribute of control unit `ctrl3` to `high`, and indicate that `ctrl3` additionally consumes the signal `s5`. Finally, after local modifications, the user can connect to the server again to upload the modified model. The collaboration server will process the modified model, and apply the detected changes to the central copy of the model - provided that they do not violate any write access control.

The previous paragraph was written under the assumption that the model was not modified on the server in the meantime. Then Pump Control Engineer would have to download the updated version of the filtered model and merge the two versions before being able to commit their changes if exists. Discussion of collaboration patterns and model merging are out of scope of this paper.

2.4 Challenges

Deriving from the goals stated in Sec. 1.1, we identify the following challenges.

C1.1 *Fine-grained Access Control.*
To meet **G1**, the solution must define and interpret a policy language that allows to permit or deny model access separately for individual model elements.

C1.2 *Context-dependent Access.*
To meet **G1**, the solution must define and interpret a policy language where access to a model asset may be granted or denied based on its attributes, relationships with other model elements, the properties of these other model elements, and in general, the wider context of the asset.

C2.1 *Model Compatibility.*
To meet **G2** in an off-line collaboration scenario, the solution must be able to present the information available to a given user as a self-contained model, in a format that can be stored, processed, displayed and edited by off-the-shelf model tooling commonly associated with the modeling language.

C2.2 *Offline Models.*
To meet **G2** in an off-line collaboration scenario, the solution must be able to present the information available to a given user as a self-contained model, that can be independently used without maintaining connectivity with any central server, peer machine, or authority responsible for access control.

C3.1 *Incrementality.*
To meet **G3** in an on-line collaboration scenario, the solution must be able to process model modifications initiated by a user and apply the consequences to the views available to other users without re-processing the unchanged parts of the model.

Challenge **C3.1** deserves some more explanation. In the online collaboration scenarios, users may edit their views of the model virtually simultaneously. If, upon a modification performed by a user, the view available to them had to be traversed entirely, and the views presented to other users had to be regenerated from scratch, collaboration on larger models would have both significant performance issues as well as low usability (since the models being used could simply disappear to be replaced by a brand-new model). Therefore *source and target incrementality* (as defined in [12]) are necessary for efficient online collaboration, implying that only the changes have to be inspected and propagated. On the other hand, in the offline scenario, incrementality is neither crucial (as access control is not applied repeatedly on every elementary model manipulation) nor easily achievable (since users upload complete model files, not just small changes).

3 Access Control of Models

3.1 Assets and Internal Consistency

In order to tackle challenges **C1.1** and **C2.1**, we first analyze how models can be decomposed into individual *assets* for which access can be permitted and denied, and under what conditions can a filtered set of such assets be represented as a model that can be processed by standard tools.

3.1.1 Model Facts as Assets

For the purposes of access control, a model is conceived as a set of elementary *model facts*. For instance, in case of RDF, model facts would be the individual triples constituting the model. These model facts would be the assets that the access control policy will protect.

For example, we decompose models in the EMF modeling platform into the following kinds of model facts:

Object facts are pairs formed of a model element (EObject) with its exact type (EClass), for each model element object; e.g. `obj(o1,Composite)`.

Reference facts are triples formed of a source EObject, a reference type (EReference) and the referenced EObject, for each containment link and cross-link between objects; e.g. `ref(o2,consumes,o12)`.

Attribute facts are triples formed of a source EObject, an attribute name (EAttribute) and the attribute value, for each (non-default) attribute value assignment; e.g. `attr(o10,cycle,low)`.

Resource facts are pairs formed of a Resource (essentially a file containing a model or a fragment of a model) and its path (actually, URI) relative to the root location of the model; e.g. `res(r1,...)`.

Root facts are pairs formed of a Resource and a root element (EObject) of the resource; e.g. `root(r1,o1)`. There is one such fact for each EObject that is not contained in other EObjects, but is rather a top-level element in its respective resource.

Note that there are multi-valued attributes and references, where an EObject is allowed to host multiple attribute values (or reference endpoints) for that property. For such properties, each of these multiple entries at a source EObject will be represented by a separate attribute (or reference) model fact. Moreover, there are *opposite* references defined as a pair of references where the existence of a relation depends on its pair. For example, the opposite of a containment reference *eContainment* is the container reference *eContainer*.

These model facts are the *assets* that the access control policy will protect; though there are a few deviations from one-to-one correspondence, such as a reference and its opposite reference (if exists) are considered a single asset.

Example In our running example, the following facts exist according to the model:

Object facts -

```
obj(root,Composite),
obj(c1,Composite),
obj(c2,Composite),
obj(ctrl1,FanControl),
obj(ctrl2,PumpControl),
obj(ctrl3,HeaterControl),
obj(ctrl4,PumpControl),
obj(s1,Signal),
    obj(s2,Signal),
    obj(s3,Signal),
    obj(s4,ConfidentalSignal),
    obj(s5,Signal),
    obj(s6,ConfidentalSignal)
```

Reference facts -

```
    ref(root,submodules,c1),
    ref(root,submodules,ctrl1),
    ref(root,submodules,ctrl2),
    ref(root,consumes,s1),
    ref(root,consumes,s2),
    ref(ctrl1,consumes,s3),
    ref(ctrl1,provides,s1),
    ref(c1,submodules,c2),
    ref(c1,submodules,ctrl3),
    ref(c1,consumes,s3),
    ref(c1,consumes,s4),
    ref(ctrl2,provides,s2),
    ref(ctrl3,provides,s3),
    ref(ctrl3,provides,s4),
    ref(c2,submodules,ctrl4),
    ref(c2,consumes,s5),
    ref(c2,consumes,s6),
    ref(ctrl4,provides,s5),
    ref(ctrl4,provides,s6)
```

Attribute facts -

```
    attr(root,vendor,A),
    attr(c1,vendor,B),
    attr(c2,vendor,C),
    attr(ctrl1,cycle,::low),
    attr(ctrl2,cycle,::low),
    attr(ctrl3,cycle,::low),
    attr(ctrl4,cycle,::low),
    attr(s1,frequency,30),
    attr(s1,documentation,Error Signal),
    attr(s2,frequency,29),
    attr(s2,documentation,Debug Signal),
    attr(s3,frequency,6),
    attr(s3,documentation,Debug Signal),
    attr(s4,frequency,31),
    attr(s4,documentation,Confidential Signal),
    attr(s5,frequency,10),
    attr(s5,documentation,Output Signal),
    attr(s6,frequency,6),
    attr(s6,documentation,Confidential Signal),
```

Root facts -

```
    res(system,root)
```

3.1.2 *Internal Consistency of Models*

An arbitrary set of model facts does not necessarily constitute a valid model; there may be *internal consistency constraints* imposed on the facts by the modeling platform to ensure the integrity of the model representation and the ability to persist, read, and traverse models. Challenge **C2.1** requires that filtered models must be synthesized as a set of model facts compatible with all internal consistency rules.

We distinguish these low-level internal consistency rules from high-level, language-specific *well-formedness constraints*. Violating the latter kind does not prevent a model from being processed and stored in the given modeling technology, as error markers can be placed on such violations. Thus only internal consistency is required for access control.

Object Existence Attributes and references imply that the objects involved exist, having a type compatible with the type of the attribute or reference.

Containment Hierarchy Objects must either be root objects of the model, or be transitively contained by a root object via a chain of objects that are all existing.

Opposite Features For reference types having an opposite, reference facts of the two types come in symmetric pairs.

Multiplicity Constraints Number of a specific references of an object needs to satisfy the multiplicity constraints.

3.2 Model Obfuscation

Obfuscation is defined as the process of *"making something less clear and harder to understand, especially intentionally"*. The first purpose of obfuscation in programming was to distribute C sources in an encrypted way to prevent the access to confidential intellectual property in the code [13]. In a modeling environment, the same concept applies.

A model obfuscator such as VIATRA Model Obfuscator [14] obfuscates structured graph-like models (e.g. XML documents, EMF-based models) by altering data values (such as names, identifiers or other strings) in a way that the structure of the model remains the same. Two data values that were identical before the obfuscation will also be identical after it, but the obfuscated value computed based on an input obfuscation string will be completely different (e.g. "root" may become "oA3DD43CF5").

Several publications [15], [16] discuss implementation of the obfuscation function which is out of scope of this paper. We assumes that an obfuscated identifier remains unique and it is possible to revert it by the original owner of the model using a private key.

In the context of access control, obfuscated data describes the existence of a model fact (e.g. a value is assigned to the attribute of an object), but the meaning of that fact remains secret. Additionally, when only the existence of object needs to be presented, its identifiers would be obfuscated while the other features are hidden (if the object is identified by a set of its attribute values e.g. in our example language Fig. 2, where each object is identified by an *id* attribute). Hence, the correspondences between the objects of the original model and its secure view remains.

3.3 Access Control Rules and Queries

To meet challenge **C1.1** stated in Sec. 2.4, our fine-grained access control policy has to be able to assign permissions separately for each model fact. Therefore the policies are constructed from a list of *access control rule*s, each of which controls the access to a selected set of model facts by certain users or groups, and may either allow or deny the read and/or write operation. Thus different rules can impose different restrictions on different model facts.

To address challenge **C1.2**, the policy must be able to take into account the wider context of the model facts when determining permissions. For this purpose, we propose to associate each access control rule with a *model query* to specify exactly which model facts the rule controls. These

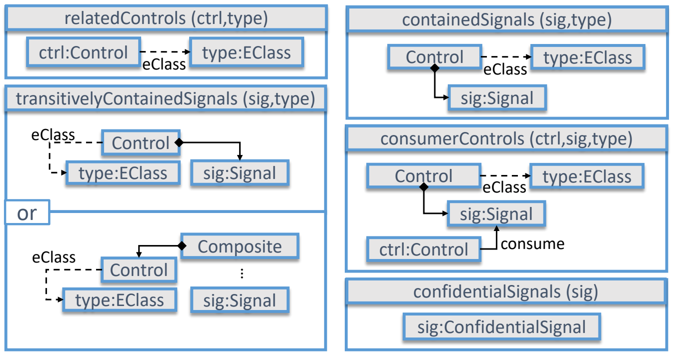


Figure 4: Graph Patterns for the Running Example

queries can identify the involved model facts, and can take into account the wider context such as properties of the object, its connections to other objects, etc.

3.3.1 *Graph Queries*

Model queries are important components in model-driven tool chains. They are widely used in MDE for specifying reports, derived features, well-formedness constraints; also rule preconditions in model transformations or design space exploration. Although model queries can be implemented using a general-purpose programming language (Java), declarative query languages may be more concise and easier to learn, among other advantages. Modeling platforms (e.g. the Eclipse Modeling Framework *(EMF)* [11]) support various query languages.

Fig. 4 displays graph patterns that specify queries against a wind turbine model. When a query is evaluated, the results consist of a set of *pattern matches*. In case of `relatedControls`, a match is a pair `<ctrl, type>` where `ctrl` is an object in the model of type `Control`, `type` is an EClass and `type` is set to the `eClass` reference of `ctrl`. In the instance model of Fig. 3, the pattern has several matches: `<ctrl1,FanControl>`, `<ctrl2,PumpControl>`, `<ctrl3,HeaterControl>` and `<ctrl4,PumpControl>`. Note that this query will be useful to determine which controls are related to which specialists (**R1**).

Graph patterns can be composed in order to reuse common query parts, and also to express disjunction, negation, and (surpassing the expressive power of first-order formulae) *transitive closure*. `transitivelyContainedSignals` selects all the `<sig,type>` pairs where (i) `sig` is contained by a `Control` object which `eClass` reference is set to `type` EClass *or* (ii) `sig` is contained by a `Composite` which also contains a `Control` object with `type` EClass. However, this query would have several matches, each parameter of the query can be bound to a specific value which filters the set of matches. In this case, if `type` parameter is bound to `HeaterControl`, the matches will be the following ones: `<s3,HeaterControl>`, `<s4,HeaterControl>`, `<s5,HeaterControl>` and `<s6,HeaterControl>`. Note that this query helps to determine which signals shall be revealed to which specialists when the `type` parameter is bound to different control types (**R2**).

For capturing access control queries over EMF models, we have chosen the EMF-IncQuery framework [17], due

Listing 1 Query Definition `containedSignals`

```
pattern containedSignals(sig: Signal,type)
{
  Control.eClass(control, type);
  Control.provides(control, sig);
}
```

to its expressive power (beyond that of first-order formulae) and incremental evaluation capabilities. The former property helps with identifying the contexts of assets (challenge **C1.2**), while the latter is necessary to meet the performance needs of the online scenario (challenge **C3.1**).

As an example, Lst. 1 defines `containedSignals` pattern in EMF-IncQuery syntax which selects all `<sig,type>` pairs where `sig` is provided by a `Control` object with `type` EClass. Note that this query aims to select the modifiable signals for the specialists (**R3**).

Rest of the patterns represented in Fig. 4 describe the following cases: `consumerControls` selects `<ctrl,sig,type>` tuples where `ctrl` consumes `sig` which is contained by another object of `Control` with `type` EClass; `confidentialSignals` queries all the signals of type `ConfidentialSignal`. Note that the former query will help us to select all controls that consumes a signal provided by a control unit related to the specialists (**R4**), while the latter query will selects confidential signals need to be hidden from the specialists (**R5**).

3.3.2 *Access Control Policy Definition*

Using graph patterns, we are able to capture access control rules to apply different accessibility level (allow, obfuscate, deny) for the operation types (read and write) of a set of assets in the name of a specific user or group of users. Lst. 2 describes the rules specific to all *Fan Control Engineer* users and shows the key concepts of the language. Lets take a more detailed look at the `pumpControl` rule. It uses the `relatedControls` query to select all controls in the model and binds the `PumpControl` EClass to the `type` parameter of the query to filter match set: {`<ctrl2,PumpControl>`, `<ctrl4,PumpControl>`}. As a projective mapping, all the substitutions of `ctrl` parameter specify object assets: `obj(ctrl2, PumpControl)`, `obj(ctrl4, PumpControl)`. The rule is applied on these assets.

Rule `accessibleSignal` grants read permissions all signals provided by a module that is either (a) a composite that directly contains a `PumpControl` object, or (b) any submodule (including `PumpControl` objects) contained *transitively* in such a composite selected by `transitivelyContainedSignals`. Rule `modifiableSignal` allows read and write operations to signals provided by `PumpControl` objects. Rule `accessibleConsumer` provides the readability of `consume` references between any type of control unit that consumes a signal provided by a `PumpControl` object. Finally rule `denyConfidentialSignal` denies the read and write operations of confidential signals from a wider group of user which contains each type of specialist (pump control engineer, heater control engineer and fan control engineer).

Listing 2 Rules specific to user *PumpControlEngineer*

```
// R1: Enable read and write operations for
// all PumpControl objects
rule pumpControl
      allow RW to PumpControlEngineer {
  select obj(ctrl)
  from query "relatedControls"
  where type bound to PumpControl
}
// R2: Enable read operation for signals
// accessible from a PumpControl
rule accessibleSignal
      allow R to PumpControlEngineer {
  select obj(sig)
  from query "transitivelyContainedSignals"
  where type bound to PumpControl
}
// R3: Enable read and write operations for
// signals contained by a PumpControl
rule modifiableSignal
      allow RW to PumpControlEngineer {
  select obj(sig)
  from query "containedSignals"
  where type bound to PumpControl
}
// R4: Enable read operation for consume
// references connected to modifiable signals
rule accessibleConsumer
      allow R to PumpControlEngineer {
  select ref(ctrl→consume→sig)
  from query "consumerControls"
  where type bound to PumpControl
}
// R5: Disable read and operation for all
// confidential signals
rule denyConfidentialSignal
      deny RW to specialists {
  select obj(sig)
  from query "confidentialSignals"
}
```

Each rule assigns *nominal permissions* to the models facts. In the following list, the nominal permissions are presented specified by each rule.

Rule `pumpControl`:
`allow RW obj(ctrl2,PumpControl),`
`allow RW obj(ctrl4,PumpControl)`

Rule `accessibleSignal`:
`allow R obj(s1,Signal),`
`allow R obj(s2,Signal),`
`allow R obj(s3,Signal),`
`allow R obj(s4,ConfidentalSignal),`
`allow R obj(s5,Signal),`
`allow R obj(s6,ConfidentalSignal)`

Rule `modifiableSignal`:
`allow RW obj(s2,Signal),`
`allow RW obj(s5,Signal),`
`allow RW obj(s6,ConfidentalSignal)`

Rule `accessibleConsumer`:
`allow R ref(root,consumes,s2),`
`allow R ref(c2,consumes,s5),`
`allow R ref(c2,consumes,s6)`

Rule `denyConfidentialSignal`:
`deny RW obj(s4,ConfidentalSignal),`

Listing 3 Policy specific to user *PumpControlEngineer*

```
policy WindTurbines deny RW by default {
 rule pumpControl ... { ... }
 rule accessibleSignal ... { ... }
 rule modifiableSignal ... { ... }
 rule accessibleConsumer ... { ... }
 rule denyConfidentialSignal ... { ... }
} with restrictive resolution
```

`deny RW obj(s6,ConfidentalSignal)`

Naturally, we cannot expect from the language engineers to define rules that cover all the facts in the models. Thus, *default* permissions need to be set at beginning of the policy definitions as it is depicted in Lst. 3. Policy definitions consist of access control rules and provide default permissions for the facts that are not covered by the rules. In our example, all the facts in the model are denied. Moreover, the rules can assign contradicting permissions for facts which has to be resolve. Hence, a conflict resolution type has to be set to calculate the *effective permissions* for each asset to resolve conflicts.

3.3.3 Effective Permissions

While access control rules directly assign *nominal permissions* to the model facts, the actually applied *effective permissions* may be different. There are four reasons for the discrepancy:

- Multiple *conflicting rules* in the policy may impose contradicting judgements on the same model fact. And in cases not covered directly by any rule, a *default* behaviour must apply.
- Demanding the *sanity* of the access control scheme leads to some implicit consequences of existing rules.
- Finally, in order to meet **C2.1**, the *internal consistency* requirements specific to the model representation / technology (see Sec. 3.1) may introduce *read and write dependencies* between the permissions of individual model facts.

There are multiple ways for resolving conflicts and inconsistencies and deriving a consistent set of effective permissions from nominal permissions. Thus the security policy itself includes a directive that selects the method to apply, so that the policy designer can select the option most fitting to the use case. In the following, we give a non-exhaustive list of such permission dependencies, and outline some reconciliation strategies as well. We plan to revisit conflict resolution in more detail in a separate paper [9].

Conflicts and defaults. In case two rules assign contradicting nominal permissions to a fact, one useful resolution strategy is to interpret the order of the rules in the list as their priority ranking; this way, one can always add a more specific rule to override a general rule in a special case. And if no rules apply to a given model fact, a sensible default may be to allow full access to it.

Sanity. The sanity of the policy implies that a user should not be allowed to write values / model facts that are not visible by them. Therefore without effective read permission, write permission is automatically denied as well, even if there are no rules to deny nominal write permissions.

Read dependencies. Effective read permissions may depend on permissions on other model facts. If a model element is not visible, its references pointing inward or outward and its attributes shall not be effectively readable either, otherwise the set of visible facts would not form a self-consistent model. In modeling platforms (such as EMF) with a notion of containment between objects, effectively visible objects cannot be contained in effectively invisible objects (as the latter do not exist in the front model); this can be solved either by inventing a new container for the orphaned object (e.g. promoting it to a top-level object of the model); or alternatively by applying the semantics that an object effectively hidden from the front model will effectively hide the entire containment subtree rooted there (this latter choice will be used in the case study).

Write dependencies. Effective write permissions likewise have dependencies. In general, creating/modifying/removing references between objects requires (in addition to the nominal permission) an effectively modifiable source object and an effectively visible target object; but some modeling platforms including EMF have bidirectional references (or opposites), for which internal consistency dictates that the target object must be effectively modifiable as well. A metamodel may constrain a reference (or attribute) to be single-valued; assigning a new target to the reference would automatically remove the old one, so a user can only be effectively allowed the former write operation if they are effectively permitted the latter. Similarly, removing an object from the model implies removing all references pointing to it, and all objects contained within it, thus yet another write permission dependency has to be introduced.

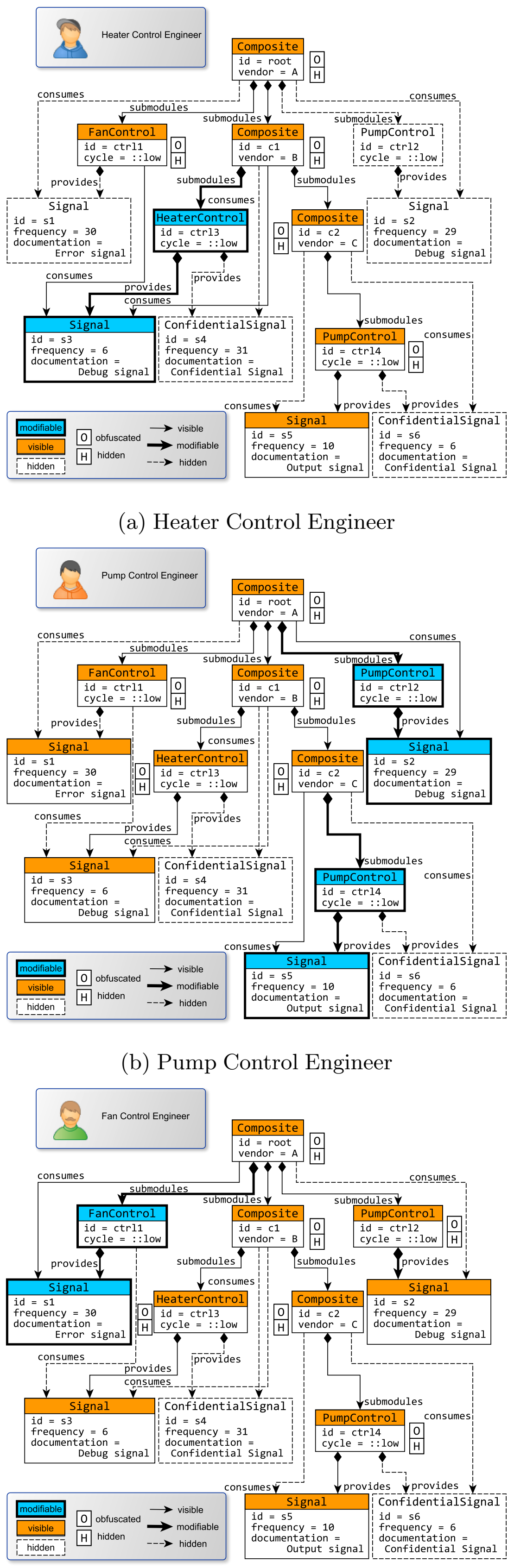
3.3.4 *Solving the Case Study*

The access control policy presented in Lst. 3 is set up to meet the security needs of the running example introduced in Sec. 2.2. A possible solution of effective permissions is visualized in Fig. 5. For instance, *Pump Control Engineer*s have full access for the PumpControl objects and their provided Signals (squares marked with bold borders and blue headers); however they cannot access to ConfidentialSignal objects (squares with dashed borders); rest of the objects are visible, but not modifiable for this group of users (squares with thick borders and organge headers). If an object is only required to preserve read dependencies, its identifier is obfuscated (marked with "O" letter in a square next to the attribute) and all other attributes remain hidden ("H" letter in the square). Finally, bold edges are modifiable by the engineers, i.e. the modifiable signals (`s2`, `s5`) can be removed from their container, or new signals can be created under the modifiable controls (`ctrl2`, `ctrl4`); thick edges represent visible references (in this example, these are required mostly to preserve containment hierarchy); and the rest of the dashed edges are hidden from the engineers.

4 Bidirectional Model Transformation for Access Control

4.1 The Access Control Lens

Due to read access control, some users are not allowed to learn certain model facts. This means that the complete model (which we will refer to as the *gold* model) differs



(a) Heater Control Engineer

(b) Pump Control Engineer

(c) Fan Control Engineer

Figure 5: Effective Permissions of the Example

from the view of the complete model that is exposed to a particular user (the *front* model).

In theory, access control could be implemented without manifesting the front model, by hiding the entire gold model behind a model access layer that is aware of the security policy and enforces access control rules upon each read and write operation performed by the user. However, our stated challenge **C2.1** requires users to be able to access their front models using standard modeling tools; moreover, challenge **C2.2** requires that in the offline collaboration scenario, they can "take home" their front model files without being directly connected to the gold model. In order to meet these goals, we propose to manifest the front models of users as standalone models, derived from a corresponding gold model by applying a *model transformation*.

A central concern is to keep the front models of users aware of changes performed by other users (as long as the access control policy allows revealing this information). This means that changes performed by one user on their front model must be propagated (potentially in filtered form) to the front models of other users. For conceptual simplicity, this is always performed by propagating information through the gold model. Thus the *synchronization chain* is as follows: when a user modifies their front model, the gold model is adjusted accordingly; then the front model of every other user is updated to reflect the modified gold model. The goal of providing change propagation in general is thus reduced to the simpler task of propagating changes between a single front and gold model pair.

In the literature of bidirectional transformations [18], a *lens* (or view-update) is defined as an asymmetric bidirectional transformations relationship where a source knowledge base completely determines a derived (view) knowledge base, while the latter may not contain all information contained in the former, but can still be updated directly. The two operations that have crucial importance in realizing a lens relationship are the following:

- GET, which obtains the derived knowledge base from the source knowledge base that completely determines it, and
- PUTBACK, which updates the source knowledge base, based on the derived view and the previous version of the source (the latter is required as the derived view does not contain all information).

The kind of bidirectional transformation relationship we find between a gold model (containing all facts) and a front model (containing a filtered view) fits the definition of a lens. The GET operation applies the access control policy for filtering the gold model into the front model. The PUTBACK operation takes a front model updated by the user, and transfers the changes back into the gold model.

The lens concept is illustrated by Fig. 6. Initially, the GET operation is carried out to obtain the front model for a given user from the gold model. Due to the read access control rules, some objects in the model may be hidden (along with their connections to other objects); additionally, some connections between otherwise visible objects may be hidden as well; finally, some attribute values of visible objects may be omitted, obfuscated, or hidden altogether. If the user subsequently updates the front model, the PUTBACK operation checks whether these modifications were allowed

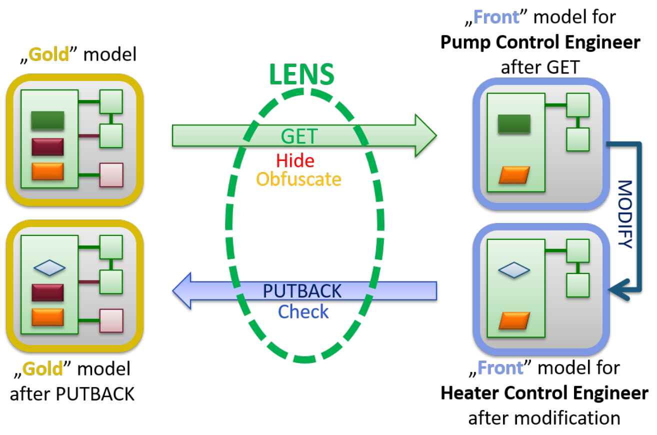


Figure 6: The MONDO Security Lens: Full Circle

by the write access control rules. If yes, the changes are propagated back to the gold model, keeping those model elements that were hidden from the user intact (preserved from the previous version of the gold model).

Write access control checks are performed by the PUTBACK operation for the following reason. Write access rules (a) may prevent a user from writing to the model, and (b) they are defined based on queries that are to be evaluated on the model. Since only the gold model contains all information, such queries cannot be evaluated (at best, they can be approximated) directly on the front repository of each user, and thus write access control can only be enforced by taking into account the gold model as well. Therefore, write access control must be combined with the lens transformation. In particular, PUTBACK must check write permissions; and fail (rolling back any effects of the commit or operation) if applying the modification to the gold model is not allowed.

The proposed lens can be used to realize the synchronization chain outlined above: when a user modifies their front model, the PUTBACK operation is invoked to adjust the gold model; then the front model of every other user is updated by GET to reflect the modified gold model. The scope of Sec. 4.2 is to define such a lens transformation between gold and front models, by specifying how GET and PUTBACK are performed to achieve read and write access control.

4.2 Transformation Design

Both GET and PUTBACK are designed as rule-based model transformations. To address challenge **C3.1**, they can be executed as incremental computations in the on-line collaboration scenario. For EMF in particular, instead of approaches specifically designed for easy specification of bidirectional transformations, we chose the unidirectional but reactive VIATRA framework for target-incremental transformations, with EMF-INCQUERY for source-incremental model queries. Note that adopters merely have to write a policy, and we provide the transformations that interpret them; so conciseness of the transformation program is not an issue.

VIATRA transformations can be specified using *transformation rules*. A rule is associated with a graph query *precondition*, an *action* (parametrized by a match of the precondition pattern), and a numerical *priority* value. Transformation execution involves repeatedly executing (*firing*) rules: finding the matches of rule preconditions of all rules (this set of matches is efficiently and incrementally maintained during the transformation), selecting a match that belongs to the rule with the lowest priority value, and executing the action of the rule on that match; the loop terminates when there are no more precondition matches.

We distinguish four *groups of transformation rules*: both GET and PUTBACK have one group each for adding model facts to the target model if a corresponding fact is present in the source model (*additive rules*), and one group each for removing model facts from the target model if no corresponding fact is present in the source model (*subtractive rules*). All four groups consist of one rule for each kind of model fact; in case of EMF, we distinguish 5 kinds of model facts (see Sec. 3.1.1); this makes twenty EMF transformation rules altogether.

To address challenge **C2.1**, the transformation has to always end up with target models that have internal consistency. This aim is supported by the reconciliation strategy that derives the effective permissions (see Sec. 3.3.3) from nominal permissions. According to a given strategy, effective permissions are expressed as composite model queries, taking into account the queries associated with access control rules, as well as the dependencies between model facts. The computation of effective permissions is executed incrementally by the query engine.

Based on such a set of effective permissions, model queries can be applied to obtain the effectively visible part of the gold model; and then the job of the transformation is more or less reduced to simply synchronizing changed model facts between the front model and the visible gold model. Thus the transformation rules themselves are almost trivial; still, there are a few ways in which they are more complicated than just verbatim copying:

- Model objects of the two models reside at different memory addresses, so the transformation must set up and maintain a one-to-one mapping (the binary relation called *object correspondence*), that can be used to translate model facts when propagating changes.
- In case of obfuscation features, they require translating type names or attribute values.
- The priorities of transformation rules must be chosen such that the order of rule execution takes into account the valid manipulation operations allowed by the modeling platform (e.g. can't set the attribute value before creating the object first).
- Write permissions have to be enforced by PUTBACK. As executing a rule would change the gold model, care should be taken in choosing *when* the effective permissions are consulted. The nominal permission for writing a model fact is evaluated after performing fact creation, but before performing fact deletion. The reason for this discrepancy is that, by convention, policy queries for write access control are expected to be evaluated in a state when the associated model fact exists in the model.

4.3 Identify Correspondence Relation

In case of offline collaboration, the consequence of *C2.2* challenge is that the approach requires to initialize object correspondence between the front and the gold models to execute GET and PUTBACK transformation as they are self-contained models. The challenge is identifying which elements in the front model correspond to which elements of the target model.

For example, imagine the following scenario:

1) There is a gold model M containing objects S and T.
2) GET is invoked on the gold model M to obtain a filtered and obfuscated front model m, which contains objects s and t (some of their attributes may be missing or obfuscated).
3) During GET, the EObject correspondence (see Sec. 4.2) is established, containing pairs (S, s) and (T, t).
4) The user creates a reference link from s to t in their front model and persist the modified front model as m', containing objects s' and t'.
5) PUTBACK is initiated in order to propagate the model update back to the gold model, resulting in the updated gold model M'.

Let us examine the last step in more detail. PUTBACK involves finding the gold model elements S' and T' that correspond to the source and target objects s', t' of the newly created front reference, so that the gold reference from S' to T' can be created. This requires the mappings (S', s') and (T', t') to be present in the EObject correspondence.

Since we are discussing the offline case, the model m was modified offline into m', and thus m' has to be loaded anew when performing PUTBACK. Consequently, objects s', t' loaded from disk will not be guaranteed to share the memory address of the original objects s and t. Similarly, S' and T' are identical copies of, but not the same as, S and T. Note that EMF provides no built-in mechanisms to trace s', t', S', T' back to s, t, S and T, respectively. Therefore the pairs (S, s) and (T, t) in the EObject correspondence created by GET cannot be taken into account when looking for the source and target gold elements for the new reference link to be created.

As there are multiple approaches that can be used to provide the correspondences, we give a non-exhaustive list of them in the following.

Environment of Objects. Identification of correspondences shows similarities to *model comparison* or *matching* phase of model merging [19], where changes need to be identified between different version of the same models. For that purpose, several tools uses heuristics based on the structural similarities of the versions.

Soft Traceability Links. A. Hegedus et al [20] introduces the concept of query-driven soft traceability links, which let us provide links between model parts using graph queries. The advantage of the approach is that these links are evaluated on demand and both side of the links can be changed.

Identifiers. The most trivial solution is using unique identifiers for the objects in the model. EMF supports the usage of universally unique identifiers (UUID) for the objects at serialization, but it unfortunately requires the models into XMI format. Ecore modeling language also allows the language engineers to mark one or more specific attributes as identifiers which can help building correspondences.

The current state of our approach uses specific attributes to provide permanent identifiers [1]. Such a permanent identifier is preserved across model revisions and lens mappings, and can therefore be used to pre-populate the object correspondence relation. In our running example, each object has a unique *id* attribute. Note that unlike EMF, some modeling platforms (e.g. IFC [21]) automatically provide such permanent identifiers.

Returning to our previous example, note that the permanence of the identifier means that the identifier of S necessarily equals that of s (since the lens GET preserves the identifier). Also, s and s' must have the same identifier (since the identifier is immutable across model revisions), just like S and S'. In the end, it turns out that s' and S' have matching unique identifiers, and so do t' and T'. This allows PUTBACK to initialize the EObject correspondence relation with pairs (S', s') and (T', t') and then execute the transformation rules successfully.

While requiring permanent identifiers is a limitation of the approach, it is only relevant for modeling platforms that do not themselves provide this kind of traceability, and only in the offline collaboration scenario. Being able to identify model objects is a relatively low barrier for modeling languages; e.g. the original wind turbine language includes a unique identifier for all model objects.

5 Tool support for Collaborative Modeling[2]

5.1 Offline Collaboration

In the offline scenario, models are stored in a Version Control System (VCS) and users work on local working copies of the models in long transactions called commits. The goal of our approach in agreement with *C2.2* is to enforce fine-grained model access control rules on top of existing security layers available in the VCS.

In the offline case, the approach depected in Fig. 7 uses two types of repositories called *gold* and *front* depicted in Fig. 7. The *gold* repository contains complete information about the models, but it is not accessible to collaborators. Each user has a *front* repository, containing a full version history of filtered and obfuscated copies of the gold models. New model versions are first added to the front repository; then changes introduced in these revisions will be interleaved into the gold models using PUTBACK transformation; finally, the new gold revision will be propagated to the front repositories of other users using GET transformation. As a result, each collaborator continues to work with a dedicated VCS as before, unaware that this front repository may contain filtered and obfuscated information only.

However this scheme enforces the access control rules even if users access their personal front repositories using standard VCS clients and off-the-shelf modeling tools, optional client-side collaboration tools may still be used to improve user experience, e.g. for smart model merging [22], user-friendly lock management [23], or preemptive warning about potential write access violations that greatly enhances the usability and applicability of the offline scenario.

5.1.1 *Higher-level Security Lens.*

Common off-the-shelf VCS tools track the revisions of a collection of files (organized into folder). File versions coexisting in a given revision can be checked out into a *working directory* to manifest them in the file system of a computer.

Given a gold working directory containing models with complete information (among other files) and a front working directory intended for holding filtered and obfuscated information, bidirectional synchronization can be performed between the two. Since the front working directory can be derived from the gold working directory, but not the other way around, this is once again a lens relationship The GET and PUTBACK operations of the working directory-level lens are implemented by (a) calling the respective transformation of the previously introduced model-level security lens (see Sec. 4.2) on each access controlled model found in the working directory, and (b) simply copying miscellaneous files that are not part of models, without modifying their contents.

The working directory-level security lens can be further lifted on to the level of entire repository revision histories. The entire revision graph of the gold repository can be mapped to an isomorphic revision graph of the front repository. The repository-level security lens has GET to propagate new revisions in the gold repository to their filtered and obfuscated counterparts at an identical position in the revision history of the front repository. The repository-level PUTBACK will, given that write permissions are not violated, transform new revisions in the front model to gold model revisions that represent the same changes relative to their respective parent revisions.

5.1.2 *Technical Realization*

Our prototype is realized by extending an off-the-shelf VCS server (Subversion in particular, but Git integration is also planned as future work) by pre- and post-commit hook scripts that enforce access control and maintain the lens relationship and post-(un)lock hooks to ensure file level locks will not be violated in any front repository.

The design of the Collaboration Server consists of a central gold repository instance, and a separate front repository instance for each user (in some VCS systems, it may be possible to implement the same setup as different compartments within a single repository instance). Existing access control mechanisms (such as firewalls) are used to ensure that the gold model is accessible to superusers only, and each regular user can only access their own front repository. Given the address of their personal front repository, regular users can apply any compatible VCS client to communicate with their front repository, unaware of collaboration mechanisms in the background.

The default behavior of the VCS server is extended by *hook scripts*. They process VCS commits to enforce read and write access control. The front repositories are equipped with front specific pre-commit hooks, while a gold specific post-commit hook is registered at the gold repository. For performance reasons, the actual model-level *lens transformation* described in Sec. 4 is carried out by a *daemon*

1. Furthermore, access control rules must not deny read access to this identifier of an object (obfuscation is possible), unless by denying read access to the object altogether.

2. Source codes and more details are at https://github.com/FTSRG/mondo-collab-framework

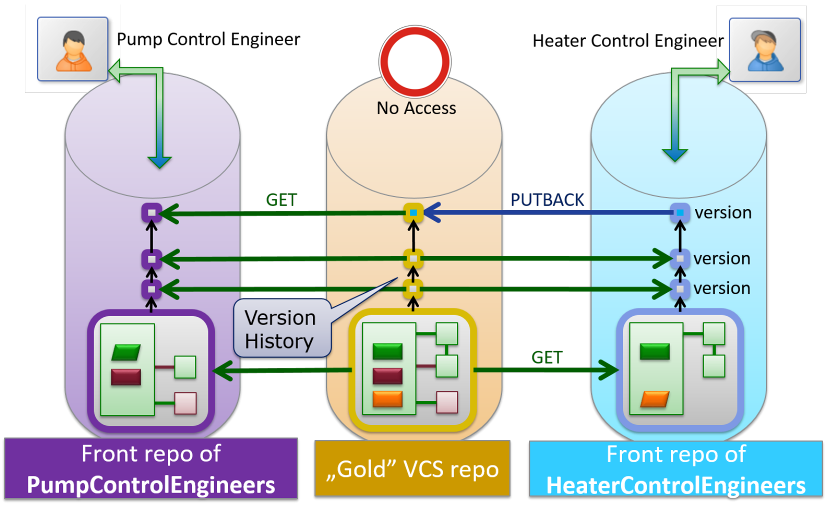


Figure 7: The MONDO Offline Collaboration Server: Architecture

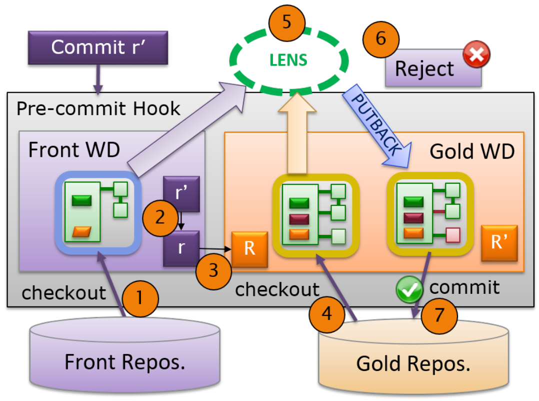


Figure 8: Front-specific Pre-commit Hook

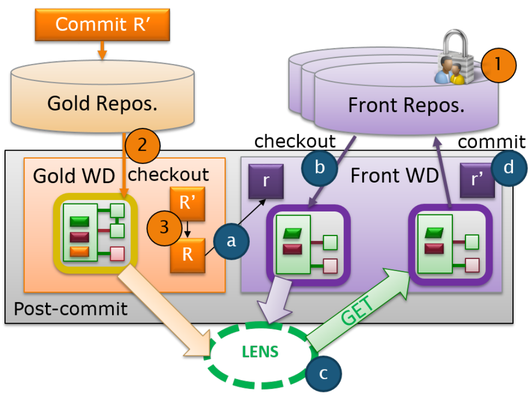


Figure 9: Gold-specific Post-commit Hook

service, which is constantly loaded in server memory. The hook scripts delegate the actual processing of model files to this daemon, so that the Java class libraries for parsing the models and realizing the transformation do not have to be re-loaded upon each hook execution.

When a user attempts to commit a new revision to their front repository, the attached pre-commit hook will be automatically invoked by the VCS server. The hook will perform the following steps (Fig. 8):

1) The contents of the new revision r' are checked out into a working directory in the file system of the server.
2) The parent revision r of the commit is identified.
3) The front revision r is traced to the corresponding revision R in the gold repository.
4) The revision R is checked out from the gold repository to another working directory.
5) The working directory-level PUTBACK transformation (see Sec. 5.1.1) is invoked to update R to R', in order to reflect the changes performed in the new commit.
6) If the PUTBACK detects any attempts by the user to perform model modifications they are not authorized to (either due to write access control or locks, see Sec. 4), then the commit process to the front repository is aborted, and the user receives an error message identifying the unauthorized modification that was attempted.
7) Otherwise, the commit is deemed successful, and R' is committed to the gold repository (with metadata such as committer name and commit message copied over from the original front repository commit).
8) Working directories and temporary files are cleaned up when the hook script terminates (regardless of success).

The gold repository may be written by people with superuser privileges or the pre-commit VCS hook described above. When a new commit is added to the gold repository, the attached post-commit hook will be automatically invoked by the VCS server. The hook will execute a new process asynchronous and returns successfully. The new process will synchronize the front repositories with the gold repository as follows (Fig. 9):

1) The process locks all the gold and front repositories in the name of an internal user to prevent new commits until the synchronization finishes.
2) The contents of the new gold revision R' are checked out into a working directory in the file system of the server.
3) The parent revision R of the commit is identified.
4) For each front repository associated with the gold repos-

itory, except for the one whose commit is being propagated right now (if the commit to the gold repository is initiated by the pre-commit hook of a front repository)

a) The gold revision R is traced to the corresponding front revision r in the gold repository.
b) The revision r is checked out from the front repository to another working directory.
c) The working directory-level Get transformation (see Sec. 5.1.1) is invoked to update r to r', in order to reflect the changes performed in the new commit.
d) The new revision r' is committed to the gold repository (with metadata such as committer name and commit message copied over from the original front repository commit).

5) Working directories, temporary files and locks are cleaned up when the the process terminates.

To prevent file-level lock violation in VCS, post-lock and post-unlock hooks are attached to the front repositories. These hooks are responsible for propagating lock specific events to the gold repository. After a front user locks a file in his/her front repository the registered post-(un)lock hook will be automatically invoked by the VCS server. The hooks will perform the following steps:

1) Identify the (un)locked file f in the front repository.
2) The file f is traced to the corresponding file F in the gold repository.
3) The file F is (un)locked in the name of an internal user.

Note that users get the response of their commit right after the collaboration server tried to propagate back the changes to gold repository. If any access control rule is violated the front-specific pre-commit hook fails. At the last phase of pre-commit, the VCS declines the commit action to the gold repository, if any modified files are locked on the gold repository. Hence, the hook fails again and prevent the VCS specific file level locks. On the other hand, if everything goes well the users do not need to wait for synchronizing with the remaining front repositories.

On-demand synchronization would improve our solution, where the change propagation to front repositories would be requested upon the execution of a *checkout* or *update* actions using pre-checkout and pre-update hooks. Unfortunately, SVN does not support these hooks, but our approach could be easily adapted to other VCSs (like Git) that supports these hooks.

5.1.3 Authorization Files

We have taken the design decision that the *authorization files* are stored and versioned in the same VCS as the models. Thus policy files may evolve naturally along with the evolution of the contents of the repository.

Policy files are writable by superusers only, but readable by every user; this means that offline clients may evaluate security rules on their offline copies themselves. Note that we do not believe that this openness of the security policy causes major security concerns, as security by obscurity is not good security; in any way, security rule names and parameters should not themselves contain sensitive design information.

5.2 Online Collaboration

In the online scenario, several users can simultaneously display and edit the same model with short transactions by using a web-based modeling tool where changes are propagated immediately to other users during *collaborative modeling sessions*. In contrast to the offline scenario, where users manipulated local copies of the models, models are kept in a server memory and users access the model directly on the server. The goal of our approach in agreement with *C3.1* is to incrementally enforce fine-grained model access control rules and on-the-fly change propagation between view models of different users.

5.2.1 Technical Realization

During a collaborative modeling session, a model kept in server memory for remote access may also be called a *whiteboard* depicted in Fig. 10. The collaboration server hosts a number of whiteboard sessions, each equipped with a gold model. Each user connected to a whiteboard is presented with their own front model, connected to the gold model via a lens relationship. The front models are initially created using Get. If a user modifies their front model, the changes are propagated to the gold model using PutBack, and propagated further to the other front models using Get again. In case of online collaboration, these lens operations are continuously and efficiently executed as *live transformation*, so users always see an up-to-date view of the model during the editing session.

Similar to modern collaborative editing tools (such as Google Sheets [24]), whiteboards can be operated transparently: whenever the first user attempts to open a given model, a new whiteboard is started; subsequent users opening the model will join the existing whiteboard. When all users have left, the whiteboard can be disposed. The model may be persisted periodically, or on demand ("save button"). The *session manager* component enables collaborators to start, join or leave whiteboard sessions, persist models to disk.

Models and authorization files are provided by an underlying VCS and accessed to them using standard checkout-commit commands. The advantage of this solution is that the online collaboration can cooperate with the offline scenario. Thus the models are persisted to the VCS using commit commands. However, if file-level conflicts occur on the level of underlying VCS, they would be need specific user interfaces to resolve them. Instead of that solution, we decided that when a new whiteboard session is initiated, file-level locks are placed on the resources of the related model to prevent conflicts in the VCS upon persisting.

5.2.2 Prototype User Interface

An initial user interface is implemented as a proof-of-concept, depicted in Fig. 11, that uses the editors automatically generated from EMF metamodels[3] which also provides similar modeling environment as the desktop Eclipse IDE. By default, it provides (1) *Tree editors* for accessing the models and (2) *Properties* view to modify the attributes of model element. We also implemented several additional views such as (3) *Current User* which shows the currently logged in user, (4) *Model Explorer* that lists the accessible models, policy definitions and related query files in the underlying VCS; (5)

3. EMF and RAP integration: https://wiki.eclipse.org/RAP/EMF_Integration

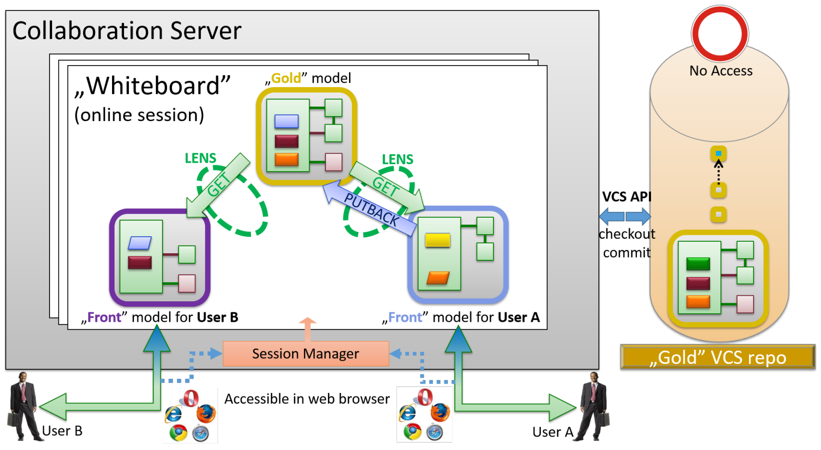


Figure 10: Overview of Online Collaboration

Whiteboard Chat provides a chat function between users in the same session; and finally the changes on the models are also tracked in the (6) *Model Log*.

However, this prototype tool is fully enough to demonstrate the capability of our query-based access control approach, we strongly believe that several other existing online collaboration tools such as GenMyModel [25] can easily adapt our solution.

6 Evaluation

Throughout the previous sections, we have demonstrated how the functional requirements stated in Sec. 2.4 are met by the proposed solution. However, as challenge **C3.1** is about performance, Sec. 6.1 provides experimental measurements to support the claim. Finally, Sec. 6.2 will discuss limitations to the presented solution.

6.1 Experimental Performance Evaluation

6.1.1 Measurement Setup

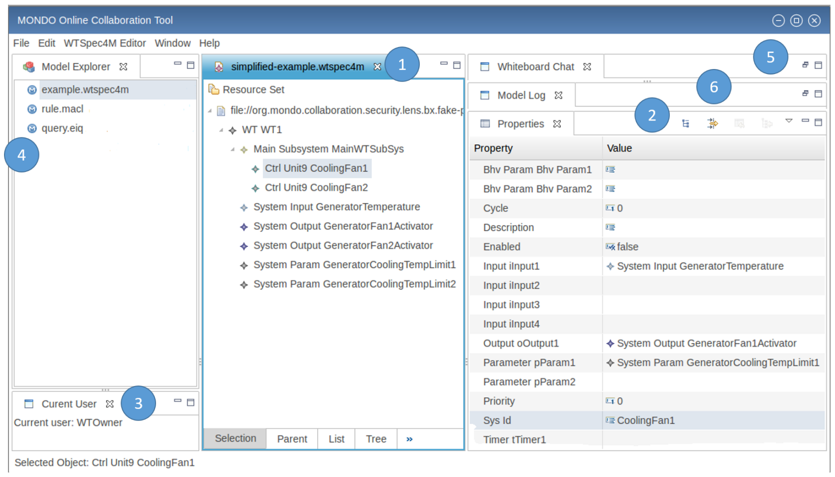
The measurement targets scenarios with a large number of users and a large access control policy. Therefore we used the metamodel of Fig. 2 with a slight modification: the control unit attribute `type` was changed from an enumeration to a string, with K different permitted values. The corresponding policy file is similar to the extract shown by Lst. 3, with one specialist engineer for each control unit type (each having two access control rules dedicated to them) and an additional principal engineer user. This means altogether $K+1$ users, 1 group and $2K+3$ access control rules.

Measurements were performed with gold instance models of various sizes. The model of size M contains a root `Composite` object, which contains M copies of the structure depicted by Fig. 3. This means $1+3M$ composite modules, $4M$ control units, $16M$ signals and $8M$ consumes cross-references. The copies are not completely identical: the `vendor` attributes are set to a different value in each copy; and the `protectedIP` attribute of composites as well as the `type` and `cycle` attributes of control units were chosen randomly from their respective ranges with uniform distribution. However, care was taken that all control unit types must occur at least once; this also implies $4M \geq K$.

The measurement was performed with $U \leq K$ specialist users and the principal engineer present (so $U+1$ front models in total).

Online case. The measurement focuses on fine-grained change propagation. To test the incremental behaviour of the lens transformation, we measured the time it took the principal engineer to perform a complex model manipulation operation on her front model, and to have the changes propagated to the front models of all users who can see it. The measured complex operation is a *signal reversal*, which changes the direction of a communication channel to the opposite. Given a random preselected signal that is currently provided by module a and consumed by module b, the reversal of this signal changes the model so that the signal is now provided by b instead of a, and consumed by a instead of b. We have selected this representative operation since (a) it involves adding and removing cross-references and a rearrangement of the containment hierarchy; (b) it does not change the size of the model, thus introduces no bias of this kind; (c) any user that can see at least one of the involved modules can see at least some aspect of the change in their front model; and (d) every access control rule in the policy (except for hiding the vendor attribute) plays a role in determining the impact of the change.

Offline case. The measurement focuses on scalability performance and the additional response time introduced by the runtime of the lens transformation. We measured the time it took specific specialist to perform several number of complex model manipulation operation on her front model, and to have the changes propagated to the gold model and from the gold model to the remaining front models of all users who can see it. The measured complex operation is a *signal addition*, which adds a new signal under the root object. We have selected this representative operation since (a) it provides that any number of new changes can be introduced into the model; (b) however, it increases the model size, but always the same ; (c) all the users can see the change in their front model; and (d) every access control rule in the policy (except for hiding the vendor attribute) plays a role in determining the impact of the change.


MONDO Online Collaboration Tool
File Edit WTSpec4M Editor Window Help
Model Explorer
simplified-example.wtspec4m
Whiteboard Chat
Model Log
Properties
Curent User
Selection Parent List Tree »
1
2
3
4
5
6

example.wtspec4m
rule.macl
query.eiq

Current user: WTOwner

Resource Set
file://org.mondo.collaboration.security.lens.bx.fake-p
WT WT1
Main Subsystem MainWTSubSys
Ctrl Unit9 CoolingFan1
Ctrl Unit9 CoolingFan2
System Input GeneratorTemperature
System Output GeneratorFan1Activator
System Output GeneratorFan2Activator
System Param GeneratorCoolingTempLimit1
System Param GeneratorCoolingTempLimit2

Property	Value
Bhv Param Bhv Param1	
Bhv Param Bhv Param2	
Cycle	0
Description	
Enabled	false
Input iInput1	System Input GeneratorTemperature
Input iInput2	
Input iInput3	
Input iInput4	
Output oOutput1	System Output GeneratorFan1Activator
Parameter pParam1	System Param GeneratorCoolingTempLimit1
Parameter pParam2	
Priority	0
Sys Id	CoolingFan1
Timer tTimer1	

Selected Object: Ctrl Unit9 CoolingFan1

Figure 11: User Interface for Online Collaboration Prototype

6.1.2 *Measurements*

We measured[4] the change propagation on a personal computer[5] with maximum a 7GB of RAM.

Online case. Two series of measurements were carried out: in the *model size scalability* series, we used $K = 50$ control unit types and $U = 10$ present users, with the size of the model ranging from $M = 25$ to $M = 350$ (8051 objects, 10850 references). In the *concurrent users scalability* series, we used $K = 100$ control unit types and the model of size $M = 100$ (2301 objects, 3100 references), with the number of specialist users joining the session r1anging from $U = 2$ to $U = 100$. For accuracy, 100 reversal operations were carried out and their execution times averaged in a single run; we have plotted the mean execution time of 10 runs, with 1 standard deviation error bars.

The results of the first series is shown by Fig. 12. No clear trend is visible on the chart (except for random fluctuations evening out on larger models). The cost of performing a single signal reversal model manipulation is low, and seems independent from the model size. This confirms that we have achieved incrementality, where computation cost is dependent on the extent of the change, but not on the size of unchanged parts of the model.

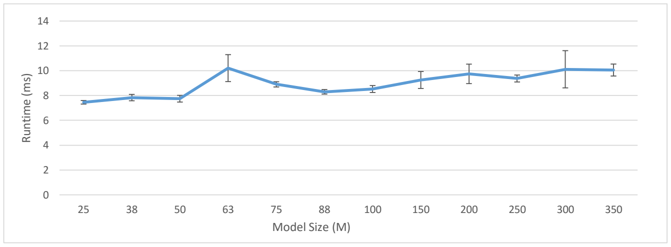


Figure 12: Average Execution Time of a Signal Reversal (varying model size)

4. Raw data and reproduction instructions at http://tinyurl.com/tse17-access-control

5. CPU: Intel Core i7-4700MQ@2.40GHz, MEM: 8GB, OS: Windows 10

The results of the second series is shown by Fig. 13. It is apparent that when very few users join the session, most signal reversals are not visible to any user other than the principal engineer; but as more and more specialist users join the session, the number of concurrent users starts to dominate the cost of model manipulation. Asymptotically, the cost of model manipulation appears proportional to the average number of front models it is propagated to.

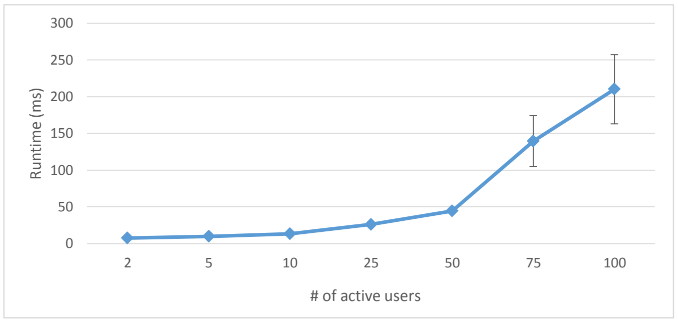


Figure 13: Average Execution Time of a Signal Reversal (varying # of concurrent users)

Offline case. Two series of measurements were carried out: in the *model size scalability* series, we used $Fr = 100$ control unit types, $Fr = 20$ front repositories and $Ch = 10$ introduced signal addition changes, with the size of the model ranging from $M = 50$ to $M = 3000$ (more than 1 billion objects and references). In the *number of front repository scalability* series, we used $K = 100$ control unit types, the model of size $M = 400$ and $Ch = 10$ introduced changes, with the number of front repositories where these changes need to propagate ranging from $Fr = 5$ to $Fr = 100$. In the *change size scalability* series, we used $K = 100$ control unit types, the model of size $M = 400$ and $Fr = 20$ front repositories, with the number of introduced changes ranging from $Ch = 10$ to $Ch = 1000$.

The charts represents the full time of the transformation including the following tasks: loading the EMF models, initializing the lens by building the correspondence tables, loading the additional files such as rules and queries, executing the transformation and finally serializing the results as a commitable new version of the models. Lower part of the bars (with *blue* color) represents the PUTBACK part of the transformation after the users get their response from the server and can continue their work on the models. Upper part of the bars (with *orange* color) visualize propagation time of the changes to synchronize with the rest of the front repositories.

The results of the first series is shown by Fig. 14. In case of the largest model, users had to wait at most 10 seconds to commit their changes in addition to the default execution time of a commit in version control system. We truly believe that this result is acceptable from the side of the users. The results also show that the response time and model size are linearly proportional, however the synchronization time is exponential.

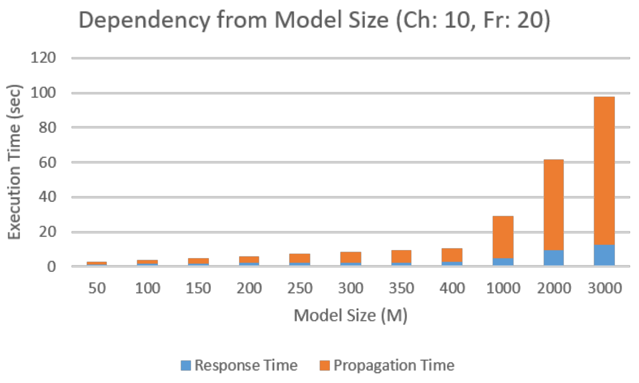


Figure 14: Average Execution Time of a Signal Addition (varying model size)

The results of the second series is shown by Fig. 15. In case of our special signal addition change, all the front repositories had to be updated to propagate the changes and the same number of modification had to be executed on those front models which declare a worts case scenario. According to these notes, it is clearly apparent on results, the number of front repository and the execution time are linearly proportional.

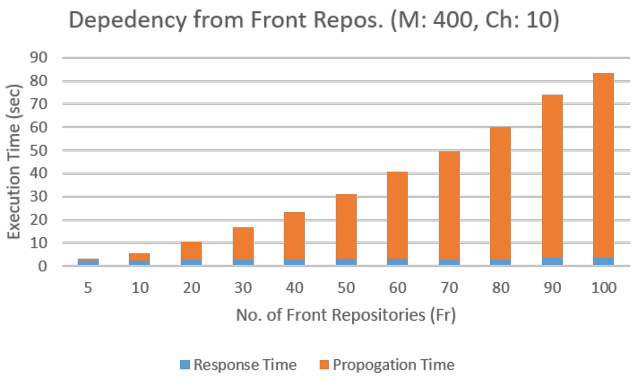


Figure 15: Average Execution Time of a Signal Addition (varying # of front repositories)

The results of the third series is shown by Fig. 15. As we can see in the results, loading the models and building the correspondence table dominate the execution time in case of small changes, thus there is no clear trend in the beginning of the chart. However it is also apparent that large size of changes (e.g. from 1000) and the execution time are linearly proportional.

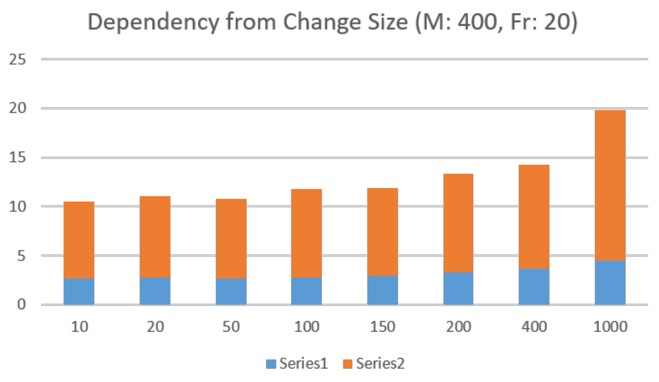


Figure 16: Average Execution Time of a Signal Addition (varying # of changes)

6.2 Assumptions and Limitations

6.2.1 User Experience for Write Access Control

User experience benefits if there is continuous feedback on the viability of the modifications attempted by the user. Advanced modeling tools may even incorporate this information into their model notation. (Obviously rejected write attempts offer a *side channel* through which some information on the hidden parts of the gold model may be gleaned. It is advised the policy designers take this into account.)

In the offline case, the server can reject (and must reject) the modifications only when the user finally submits them. A tighter feedback loop would require client-side approximation of the policy queries based on the incomplete information in the front model; this is left as future work.

In the online case, however, PUTBACK is a live transformation, and can immediately reject offending changes.

6.2.2 Ordered Lists

In EMF, some multi-valued references and attributes are *ordered* lists. Model facts introduced in Sec. 3.1 collectively represent all knowledge contained in an EMF model, with the exception of ordering information; thus the proposed solution does not respect ordering. The core reason is that there is no unique way to provide PUTBACK for ordered lists that have been filtered; therefore such a lens would necessarily violate *undoability* [26], a common requirement against bidirectional transformations. Finding an acceptable resolution of the problem (e.g. imposing a limitation that, for each user, ordered lists must be read-only unless entirely visible) is left as future work. For now, the proposed solution works properly for unordered collections.

6.2.3 Central Authority

Note that both **G2** and **G3** employ a central repository (owned by e.g. a system integrator) where the entire model is available. In a more general case, no single entity would be in possession of complete knowledge. There is an algebra [18] for combining lens transformations in various ways, suggesting a promising path for addressing this issue in future research. However, such a distributed scenario is out of scope for this paper; we address the centralized case, which is by far the most common in access control approaches.

7 Related Work

7.1 Access control in RDF triple/quad stores.

Graph-based access control is a popular strategy for many triple and quad stores (4store [27], Virtuoso , IBM DB2) developed for storing large RDF data. User privileges can be granted to for each named graph while access control is actually checked when issuing a SPARQL query. Denial of access for a graph filters the query results obtained from this specific graph. Data access in AllegroGraph [28] can be controlled on the database or catalog level (coarse-grained) as well as on the graph and triple level (fine-grained) while Stardog only allows database-level access control.

Similarly to our approach, query-based access control is discussed in [29]; the major difference being that we apply queries in an MDE environment (this has very important implications relative to RDF, see Sec. 3.1.2), and we also provide offline collaboration.

In the Oracle Database Semantic Technologies [30], access control is carried out by default on the model (graph) level. Furthermore, it can be configured on the triple (row) level, which is implemented by query rewriting. In this case, the definition of access control policies is based on so-called match and apply (graph) patterns, where the former identifies the type of access restriction while the latter injects access-control specific constraints to the query.

Another access control technique is called *label based security*, which offers (1) triple-level control using (a hierarchy of) sensitivity labels attached to each triple, and (2) RDF resource-level access control for subject/predicate/object. Explicit data access labels are implemented in [30] and are generalized into abstract tokens and operators in [31].

Context-dependent access control [32], [33] aims to filter the query result by rewriting the queries [34] in accordance with rule and graph pattern based policy specifications. A similar pattern based policy definition is complemented with precise default semantics and access conflict resolution in [35].Post-filtering query results to enforce the access control policy is also possible [36] but this strategy may have performance issues without dedicated support.

7.2 Access control in collaborative modeling environments.

Traditional version control systems (like CVS, SVN) adopt file-level access policies, which are clearly insufficient for fine-grained access control specifications. CDO [2] allows for role-based access control with type-specific (class, package and resource-level) permissions, but disallows instance level access control policy specifications. Access control is not considered in recent collaborative modeling environments like VirtualEMF [37], WebGME [38], or the tools developed according to [39].

AToMPM [40] provides fine-grained role-based access control for online collaboration; no offline scenario or query-based security is supported, though. Access control is provided at elementary manipulation level (RESTful services) in the online collaboration solution of [41].

The VehicleFORGE collaborative hardware design platform has an access control scheme TrustForge [42] that is very flexible in determining the range of users that can access a resource, but offers no query-based identification of fine-grained assets.

7.3 Model-driven security.

Model-based techniques have also been used for access control purposes. In [43], similarly to our solution, access control is enforced at runtime by program code that has been automatically generated from a model-based specification, which captures both system and security policy descriptions. This technique can provide runtime checks only on single entities by using the guarded object design pattern. A similar approach is suggested by [44], which specifies access control policies by OCL. Although this idea enables the formulation of queries that involve several objects, the efficient checking of these complex structural queries highly depends on the algorithmic experience of the system designer due to the fact that OCL handles model navigation in an imperative style, in contrast to declarative graph patterns, where several sophisticated pattern matching algorithms are readily available.

The book chapter [45] about Model-driven Security provides a detailed survey of a wide range of MDE approaches for designing secure systems, but does not cover the security of the MDE process itself.

7.4 Access Control and Bidirectional Programming.

Bidirectional Programming (BP) is an approach for defining lenses concisely, e.g. by only specifying one of GET and PUTBACK, and deriving the other. Such lenses can be directly applied for read filtering. However, [46] demonstrates that conventional BP is not sufficient for write access control. It also proposes such an integrity-preserving BP approach, focusing on string transformations (and therefore not directly applicable in MDE). There is no notion of access control policy either, so the security engineer has to develop their own lens transformation to implement access control. However, in future work, we plan to build on the high-level correctness criteria proposed for security lenses.

Similarly to our approach, a dedicated policy is used by [47], from which a lens is automatically generated to enforce access control for XML documents. In addition to the attributes and context of the assets (XML nodes), the XQuery-based policy can take into account external (subject or context) attributes as well. As it is not an MDE approach, there is no treatment of cross-references. There is no discussion of internal consistency either (see Sec. 3.1.2), except for the containment hierarchy, which is relevant for XML as well. Finally, there is no discussion of the challenges of online and offline collaboration.

7.5 Other.

Graph or logic based, declarative specification formalisms (like Datalog programs or graph transformation) can be applied for validating access control policies [48], [49], [50]. Note that the common goal of these approaches is to prove statements on the policy itself, in contrast to our technique, which enforces the access control scheme at runtime on the underlying model.

The closest representative of this research area is [51], which exploits incremental techniques to analyze evolving role-based access control policies. It restricts the set of privileges to read and write primitives in contrast to the complete, model-based hierarchy in our solution. However, the main difference still lies in the dissimilar role that access control plays as mentioned above.

8 Conclusion and Future Work

In this paper, we aimed to uniformly enhance secure online and offline collaborative modeling frameworks by using model queries for capturing fine-grained secure access control policies. Each collaborator can access a dedicated copy of the model in accordance with read permissions of the policy. Moreover, bidirectional transformations for synchronizing changes between different collaborators and check that write permissions are also respected.

We illustrated our techniques in the context of a Wind Turbine case study from the MONDO European Project, which was also used to carry out an initial experimental evaluation to assess scalability with models of increasing size, increasing change introduced by collaborators and increasing number of collaborators. In case of online collaboration, the results were promising with close to instant propagation of changes and checking of write permissions up to 75 simultaneous collaborators. In case of offline collaboration, the results show that the response time is acceptable (less than 10 additional seconds for the largest model).

As future work, we would like to primarily address the limitations presented in Sec. 6.2, and formally characterize the assumptions under which GET and PUTBACK terminate and satisfy other correctness criteria (incl. undoability) found in bidirectional transformations literature.

References

[1] J. Whittle, J. Hutchinson, and M. Rouncefield, “The state of practice in model-driven engineering,” *IEEE Software*, vol. 31, no. 3, pp. 79 – 85, 2014.

[2] T. E. Foundation, “CDO,” http://www.eclipse.org/cdo.

[3] ——, “EMFStore,” http://www.eclipse.org/emfstore.

[4] “CAESAR Research Project,” http://store.sae.org/caesar/.

[5] Aerospace vehicle systems institute, “SAVI Research Project,” http://http://savi.avsi.aero/.

[6] Apache, “Subversion,” https://subversion.apache.org/.

[7] K. F. Fogel and M. Bar, *Open source development with CVS.* Coriolis Group Books, 2001.

[8] G. Bergmann, C. Debreceni, I. Ráth, and D. Varró, “Query-based Access Control for Secure Collaborative Modeling using Bidirectional Transformations,” in *ACM/IEEE 19th Int. Conf. on MODELS*, 2016.

[9] C. Debreceni, G. Bergmann, I. Ráth, and D. Varró, “Deriving effective permissions for modeling artifacts from fine-grained access control rules,” in *1st International Workshop on Collaborative Modelling in MDE*, ACM. Saint Malo, France: ACM, 06/2016 2016.

[10] A. Bagnato, E. Brosse, A. Sadovykh, P. Maló, S. Trujillo, X. Mendialdua, and X. De Carlos, “Flexible and scalable modelling in the mondo project: Industrial case studies.” in *XM@ MoDELS*, 2014, pp. 42–51.

[11] The Eclipse Project, “Eclipse Modeling Framework,” http://www.eclipse.org/emf/.

[12] D. Hearnden, M. Lawley, and K. Raymond, “Incremental model transformation for the evolution of model-driven systems,” in *Proc. of the 9th International Conference on Model Driven Engineering Languages and Systems*, ser. LNCS, O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, Eds., vol. 4199. Genova, Italy: Springer, October 2006, pp. 321–335.

[13] R. Jaeschke, “Encrypting c source for distribution,” *Journal of C Language Translation*, vol. 2, no. 1, pp. 71–80, 1990.

[14] VIATRA Project, “VIATRA Model Obfuscator,” https://wiki.eclipse.org/VIATRA/ModelObfuscator.

[15] D. Hofheinz, J. Malone-Lee, and M. Stam, “Obfuscation for cryptographic purposes,” in *Theory of Cryptography Conference.* Springer, 2007, pp. 214–232.

[16] S. Schrittwieser and S. Katzenbeisser, “Code obfuscation against static and dynamic reverse engineering,” in *Int. Workshop on Information Hiding.* Springer, 2011, pp. 270–284.

[17] Z. Ujhelyi, G. Bergmann, Ábel Hegedüs, Ákos Horváth, B. Izsó, I. Ráth, Z. Szatmári, and D. Varró, “EMF-IncQuery: An integrated development environment for live model queries,” *Science of Computer Programming*, no. 0, pp. –, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167642314000082

[18] Z. Diskin, “Algebraic models for bidirectional model synchronization,” in *MoDELS*, 2008, pp. 21–36.

[19] K. Altmanninger, M. Seidl, and M. Wimmer, “A survey on model versioning approaches,” *International Journal of Web Information Systems*, vol. 5, no. 3, pp. 271–304, 2009.

[20] Á. Hegedüs, Á. Horváth, I. Ráth, R. R. Starr, and D. Varró, “Query-driven soft traceability links for models,” *Software & Systems Modeling*, vol. 15, pp. 733–756, 2014. [Online]. Available: http://link.springer.com/article/10.1007/s10270-014-0436-y

[21] *ISO 16739:2013: Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries*, Int. Organization for Standardization, 04 2013.

[22] C. Debreceni, I. Ráth, D. Varró, X. De Carlos, X. Mendialdua, and S. Trujillo, “Automated model merge by design space exploration,” in *International Conference on Fundamental Approaches to Software Engineering.* Springer, 2016, pp. 104–121.

[23] M. Chechik, F. Dalpiaz, C. Debreceni, J. Horkoff, I. Ráth, R. Salay, and D. Varró, “Property-based methods for collaborative model development,” in *Joint Proc. of the 3rd Int. Workshop on the Glob. Of Modeling Lang. and the 9th Int. Workshop on Multi-Paradigm Modeling.* Citeseer, 2015, pp. 1–7.

[24] N. Conner, *Google Apps: The Missing Manual: The Missing Manual.* ” O’Reilly Media, Inc.”, 2008.

[25] Axellience, “GenMyModel,” http://www.genmymodel.com.

[26] P. Stevens, “Bidirectional model transformations in QVT: semantic issues and open questions,” *Software & Systems Modeling*, vol. 9, no. 1, pp. 7–20, 2008. [Online]. Available: http://dx.doi.org/10.1007/s10270-008-0109-9

[27] Garlik, “4store,” http://4store.org/trac/wiki/GraphAccessControl.

[28] I. Franz, “AllegroGraph,” http://franz.com/agraph/allegrograph/doc/security.html.

[29] S. Dietzold and S. Auer, “S.: Access control on RDF triple stores from a semantic wiki perspective,” in *In: Scripting for the Semantic Web Workshop at 3rd European Semantic Web Conference (ESWC*, 2006.

[30] Oracle, “Database Semantic Technologies,” http://docs.oracle.com/cd/E11882_01/appdev.112/e11828/fine_grained_acc.htm.

[31] V. Papakonstantinou, M. Michou, I. Fundulaki, G. Flouris, and G. Antoniou, “Access control for RDF graphs using abstract models,” in *17th ACM Symposium on Access Control Models and Technologies, SACMAT ’12, Newark, NJ, USA - June 20 - 22, 2012.* ACM, 2012, pp. 103–112.

[32] F. Abel, J. L. D. Coi, N. Henze, A. W. Koesling, D. Krause, and D. Olmedilla, “Enabling advanced and context-dependent access control in RDF stores,” in *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Confer-*

ence, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007, ser. LNCS, vol. 4825. Springer, 2007, pp. 1–14.

[33] W. Chen and H. Stuckenschmidt, "A model-driven approach to enable access control for ontologies," in *Business Services: Konzepte, Technologien, Anwendungen. 9. Internationale Tagung Wirtschaftsinformatik 25.-27. Februar 2009, Wien*, ser. books@ocg.at, vol. 246. Österreichische Computer Gesellschaft, 2009, pp. 663–672.

[34] M. Stonebraker and E. Wong, "Access control in a relational data base management system by query modification," in *ACM '74: Proceedings of the 1974 annual conference - Volume 1*, R. C. Brown and D. E. Glaze, Eds. New York, New York, USA: ACM, 1974, pp. 180–186.

[35] G. Flouris, I. Fundulaki, M. Michou, and G. Antoniou, "Controlling access to RDF graphs," in *Future Internet - FIS 2010 - Third Future Internet Symposium, Berlin, Germany, September 20-22, 2010. Proceedings*, ser. LNCS, vol. 6369. Springer, 2010, pp. 107–117.

[36] J. Biskup and T. Weibert, "Confidentiality policies for controlled query evaluation," in *Proc. of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, ser. LNCS, S. Barker and G.-J. Ahn, Eds., vol. 4602. Redondo Beach, California, USA: Springer, July 2007, pp. 1–13.

[37] C. Clasen, F. Jouault, and J. Cabot, "Virtualemf: A model virtualization tool," in *Advances in Conceptual Modeling. Recent Developments and New Directions - ER 2011 Workshops FP-UML, MoRE-BI, Onto-CoM, SeCoGIS, Variability@ER, WISM, Brussels, Belgium, October 31 - November 3, 2011. Proceedings*, ser. LNCS, vol. 6999. Springer, 2011, pp. 332–335.

[38] M. Maroti *et al.*, "Next Generation (Meta)Modeling: Web- and Cloud-based Collaborative Tool Infrastructure," in *8th Multi-Paradigm Modeling Workshop*, Valencia, Spain, 09/2014 2014.

[39] J. Gallardo, C. Bravo, and M. A. Redondo, "A model-driven development method for collaborative modeling tools," *J. Network and Computer Applications*, vol. 35, no. 3, pp. 1086–1105, 2012.

[40] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, V. Mierlo, and H. Ergin, "AToMPM: A Web-based Modeling Environment," MODELS 2013 Demonstrations Track, 2013.

[41] M. Farwick, B. Agreiter, J. White, S. Forster, N. Lanzanasto, and R. Breu, "A web-based collaborative metamodeling environment with secure remote model access," in *Web Engineering, 10th International Conference, ICWE 2010, Vienna, Austria, July 5-9, 2010. Proceedings*, ser. LNCS, vol. 6189. Springer, 2010, pp. 278–291.

[42] P. U. DARPA VehicleFORGE, *TrustForge: Flexible Access Control for VehicleForge.mil Collaborative Environment*, 2012. [Online]. Available: http://cps-vo.org/node/6851

[43] J. Jürjens, "Model-based run-time checking of security permissions using guarded objects," in *Proc. of the 8th International Workshop on Runtime Verification*, ser. LNCS, M. Leucker, Ed., vol. 5289. Budapest, Hungary: Springer, 2008, pp. 36–50.

[44] R. Breu, G. Popp, and M. Alam, "Model based development of access policies," *International Journal on Software Tools for Technology Transfer*, vol. 9, no. 5, pp. 457–470, 2007.

[45] L. Lucio, Q. Zhang, P. H. Nguyen, M. Amrani, J. Klein, H. Vangheluwe, and Y. L. Traon, "Advances in model-driven security," *Advances in Computers*, vol. 93, pp. 103–152, 2014. [Online]. Available: http://dx.doi.org/10.1016/B978-0-12-800162-2.00003-8

[46] J. N. Foster, B. C. Pierce, and S. Zdancewic, "Updatable security views," in *Proceedings of the 2009 22Nd IEEE Computer Security Foundations Symposium*, ser. CSF '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 60–74. [Online]. Available: http://dx.doi.org/10.1109/CSF.2009.25

[47] L. Montrieux and Z. Hu, "Towards attribute-based authorisation for bidirectional programming," in *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '15. New York, NY, USA: ACM, 2015, pp. 185–196. [Online]. Available: http://doi.acm.org/10.1145/2752952.2752963

[48] M. Koch, L. V. Mancini, and F. Parisi-Presicce, "A graph-based formalism for RBAC," *ACM Transactions on Information and System Security*, vol. 5, no. 3, pp. 71–127, August 2002.

[49] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca, "A logical framework for reasoning about access control models," *ACM Transactions on Inf. and System Security*, vol. 6, pp. 71–127, 2003.

[50] R. Thion and S. Coulondre, "Representation and reasoning on role-based access control policies with conceptual graphs," in *Proc. of the 14th International Conference on Conceptual Structures*, ser. LNCS, H. Schärfe, P. Hitzler, and P. Øhrstrøm, Eds., vol. 4068. Aalborg, Denmark: Springer, 2006, pp. 427–440.

[51] M. I. Gofman, R. Luo, J. He, Y. Zhang, and P. Yang, "Incremental information flow analysis of role based access control," in *Proc. of the 2009 International Conference on Security and Management.* Las Vegas, Nevada, USA: CSREA Press, 2009, pp. 397–403.