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

Gábor Bergmann[1,2], Csaba Debreceni[1,2], István Ráth[1] and Dániel Varró[1,2]
[1]Budapest University of Technology and Economics,
Department of Measurement and Information Systems,
1117 Budapest, Magyar tudósok krt. 2.
[2]MTA-BME Lendület Research Group on Cyber-Physical Systems
{bergmann,debreceni,rath,varro}@mit.bme.hu

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 technique for secure collaborative modeling where (1) fine-grained access control for models can be defined by model queries, and (2) 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. We illustrate the approach and carry out an initial scalability assessment using a case study of the MONDO EU project.

1. INTRODUCTION

1.1 Background and Motivation

The adoption of model driven engineering (MDE) by system integrators (like airframers or car manufacturers) has been steadily increasing in the recent years [39], 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[18], EMFStore[19], 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[1] or SAVI[3]) 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[4], CVS[16]). *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,

[*]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.

MODELS '16, October 02-07, 2016, Saint-Malo, France
© 2016 ACM. ISBN 978-1-4503-4321-3/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2976767.2976793

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 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.

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.

1.2 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,*

G2 *Secure and Versatile Offline Collaboration,*

G3 *Secure and Efficient Online Collaboration.*

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.

2. CASE STUDY AND CHALLENGES

2.1 Case Study

2.1.1 Language

Several concepts 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. The metamodel, defined in EMF [36] and depicted by Figure 1, describes how the system is modeled as modules providing and consuming signals. Modules are organized in a containment hierarchy of composite modules, ultimately containing control unit modules responsible for a given type of physical device (such as pumps, heaters or fans). Composite modules may be shipped by external vendors and may express protected IP.

A sample instance model containing a hierarchy of 3 `Composite` modules and 4 `Control` units, providing 16 `Signals` altogether, is shown on Figure 2. Boxes represent objects (with attribute values as entries within the box). Arrows with diamonds represent the containment edges, while arrows without diamonds represent cross-references.

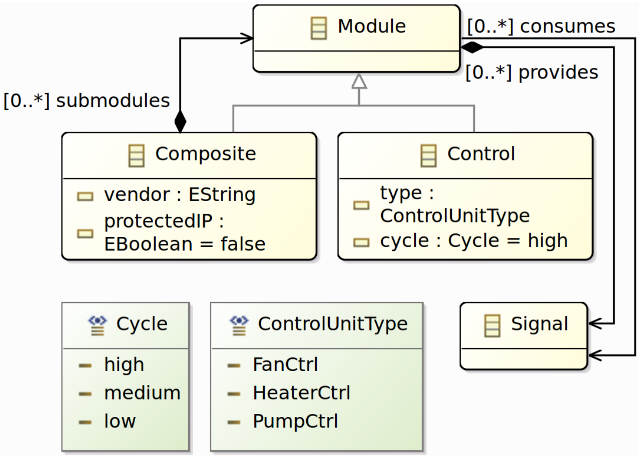

Figure 1: Simplified Metamodel of Wind Turbine Controllers

2.1.2 Access Restrictions

We assume that each control unit type (`PumpCtrl`, etc...) is associated with a specific person (referred to as *specialist*, but could also be a subcontractor or supplier) who is responsible for maintaining the model of control unit modules of that specific type. Each such user is able to modify the control units that belong to them (along with the signals provided by those modules). Additionally, they are allowed to see but not modify those composite modules (and their provided signals) that directly or indirectly contain their control units. But if the composite module is protected, its `vendor` attribute value and consumed signals is hidden (even from users who can see the consumed `Signal` object itself). Finally, there is a *principal engineer* that oversees the entire module structure and has read and write access to the entire model.

The user responsible for fan controllers (the *Fan Control Engineer*) can modify the objects marked with dark red background and white font, and can additionally see objects with blue colored label and black font. As for cross-references, the `consumes` references of `o10` are modifiable marked with thick red arrows; the reference from `o2` to `o12` is also visible to the user marked with blue dotted arrows. But the fact that `o2` consumes `o9` is not revealed, because the user is not allowed to see the `Signal o9`, as it is provided by `o7`, hidden from this user.

On the other hand, the *Pump Control Engineer*, responsible for pump controllers, can modify the objects with an red dot-dashed outline, and additionally see the objects with blue dashed outline. Thus this user would see all three composite modules, but (due to IP protection) the vendor name of `o13` would remain hidden, as well as the fact that it consumes the (otherwise visible) signal `o23`.

2.1.3 Usage Scenarios

The system integrator company is hosting the wind tur-

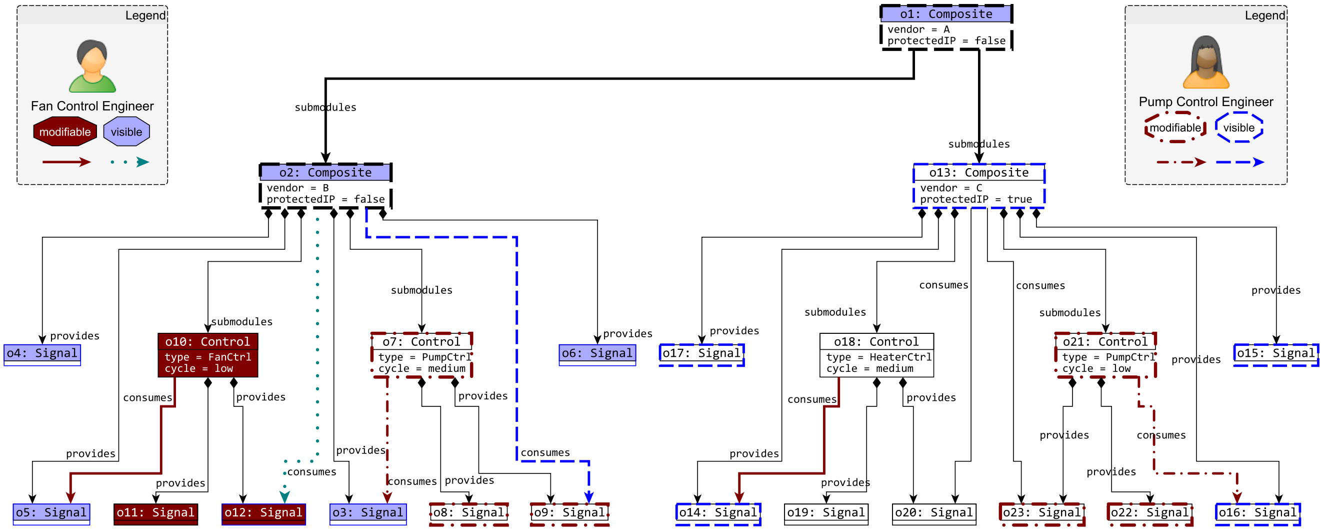

Figure 2: Sample Wind Turbine Instance Model (colored according to access privileges)

bine 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 Fan Control Engineer, this view will consist of the elements depicted on Figure 2 with blue and green shading. 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 principal engineer uses their web browser to add a new signal provided by composite module `o13`, then the Fan Control Engineer will not see this change, but the views of the Heat Control Engineer and Pump Control Engineer will immediately show the new element. Next, when the principal engineer declares that composite module `o2` consumes the previously created signal, the new cross-reference will only appear in the view of the Pump Control Engineer, the only specialist that can see both of its endpoints.

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 Fan Control Engineer can change the `cycle` attribute of control unit `o10` to `high`, and indicate that `o10` additionally consumes the signal `o3`. 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. Had the principal engineer changed the same `cycle` attribute on the server, then then Fan Control Engineer would have to download the updated version of the filtered model and merge the two versions before being able to commit. Discussion of collaboration patterns and model merging are out of scope of this paper.

2.2 Challenges

Deriving from the goals stated in Sec. 1.2, 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 [24]) 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 host multiple attribute values / reference endpoints for that property. For such properties, each of these multiple entries at a source EObject will be represented by a separate attribute resp. reference model fact.

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.

Unlike low-level internal consistency rules, violating high-level, language-specific *well-formedness constraints* would not prevent a model from *existing* in the given modeling technology, merely from being considered error-free. Thus only internal consistency is required for access control.

The following are three simple examples of EMF internal consistency constraints: (i) attribute and reference facts imply that the EObjects involved exist as an object fact, having a type compatible with the type of the attribute or reference; (ii) for reference types having an opposite, reference facts of the two types come in symmetric pairs; and (iii) reference facts of a containment type and root facts together form a *containment forest*, which contains all EObjects, and where all tree roots are resources.

3.2 Access Control Policy

3.2.1 Access Control Rules and Queries

To meet challenge **C1.1** stated in Sec. 2.2, 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 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.

Model queries are widely used in MDE for specifying reports, derived features, well-formedness constraints; also rule preconditions in model transformations or design space exploration. Modeling platforms (e.g. the Eclipse Modeling Framework *(EMF)* [36]) support various query languages.

For capturing access control queries over EMF models, we have chosen the EMF-IncQuery framework [38], due 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**).

Lst. 1 displays a graph pattern that specifies a query against a wind turbine model in the EMF-IncQuery syn-

Listing 1 Query Definition `protectedConsumes`

```
pattern protectedConsumes(
  module: Composite, signal: Signal
) {
  Module.consumes(module, signal);
  Composite.protectedIP(module, true);
}
```

Listing 2 Query Definition `objectCompositeWithType`

```
pattern objectCompositeWithType(
  composite: Composite, type
) {
  find submodules+(composite,control);
  Control.type(control, type);
}
pattern submodules(parent:Composite, child:Module){
  Composite.submodules(parent,child);
}
```

tax. When a query is evaluated, the results consist of a set of *pattern matches*. In this case, a match is a pair `<module, signal>` where `module` is an object in the model of type `Composite`, `signal` is an object of type `Signal`, `signal` is one of the signals consumed by `module`, and the `protectedIP` attribute of `module` is true. In the instance model of Figure 2, the pattern has two matches: `<o13, o20>` and `<o13, o23>`. These are exactly those cross-references that need to be hidden even from those specialists that can see both endpoints.

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*. Lst. 2 introduces a helper pattern `submodules` that expresses the relationship between a composite module and one of its submodules. This helper pattern is *transitively* embedded into `objectCompositeWithType`; thus the latter query will match pairs `<composite, type>` where `composite` is a *Composite* that contains (directly or indirectly) at least one control unit whose type attribute takes the value `type`. The pattern has the following matches in the instance model of Figure 2: `<o1, FanCtrl>`, `<o1, HeaterCtrl>`, `<o1, PumpCtrl>`, `<o2, FanCtrl>`, `<o2, PumpCtrl>`, `<o13, HeaterCtrl>`, `<o13, PumpCtrl>`. Note that this query will be useful to determine which composite modules shall be revealed to which specialists.

3.2.2 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 [11].

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.

3.2.3 Solving the Case Study

An access control policy can be set up to meet the security needs of the running example introduced in Sec. 2.1. An extract of the policy is shown in Lst. 3.

Rule `userComposite_Fan` grants read permissions to the `FanEngineer` user on `Composite` objects identified by the model query; the query uses the pattern `objectCompositeWithType` (see Lst. 2), with the variable `type` restricted to

Listing 3 Example Access Control Policy (extract)

```
// Rules specific to user FanEngineer
  //Reveal composites containing fan controllers
  rule userComposite_Fan permit R to FanEngineer {
    query "objectCompositeWithType"
    bind type value FanCtrl
  }
  //Give full access to fan control units
  rule userControl_Fan permit RW to FanEngineer {
    query "objectControlWithType"
    bind type value FanCtrl
  }
// Default rule
rule denyAllModule deny RW to specialists {
  query "objectModule"
}
// IP protected modules
rule denyProtectedConsumes deny RW to specialists {
  query "protectedConsumes"
}
rule denyProtectedVendor deny RW to specialists {
  query "protectedVendor"
}
```

the control unit type associated with this user. In effect, the user will be able to see all composite modules that contain at least one control unit module of the type FanCtrl. Similarly, the rule userControl_Fan grants additional write permission to the control units themselves. Similar rules exist for the specialists of other control unit types.

All modules not covered by the above rules will evaluate against the next rule, denyAllModule, which denied read and write access to all members of the group specialists, therefore they will not be able to see or modify any other modules. The principal engineer is not part of this group, and thus retains full access. The rule only directly affects modules, but Signals provided (contained) by hidden modules will become invisible to the given specialist as well.

Rules can also control access on the level of individual attributes and references. For instance, rule denyProtectedSignalRead hides the signals read by protected modules (see Lst. 1 for pattern protectedConsumes). An example of this would be the consumes reference from o13 to o23 (see Figure 2) hidden from the pump engineer, despite that he/she can see both the protected module and the signal.

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 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*.

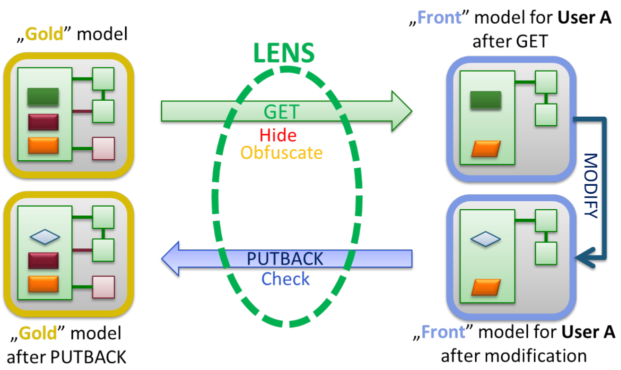

Figure 3: The MONDO Security Lens: Full Circle

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 [13], 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 Figure 3. 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. (Metamodel obfuscation and attribute value obfuscation are also available, but are out of scope for this paper.) If the user subsequently updates the front model, the PutBack operation checks whether these modifications were allowed 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 is 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. 3.2.2) 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.
- Obfuscation features are out of scope of this paper, but they also 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.

5. EVALUATION

Throughout the previous sections, we have demonstrated how the functional requirements stated in Sec. 2.2 are met by the proposed solution. However, as challenge **C3.1** is about performance, Sec. 5.1 provides experimental measurements to support the claim. Finally, Sec. 5.2 will discuss limitations to the presented solution.

5.1 Experimental Performance Evaluation

5.1.1 Measurement Setup

The measurement targets scenarios with a large number of users and a large access control policy. Therefore we used the metamodel of Figure 1 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 Figure 2. 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 at an online collaboration session (so $U + 1$ front models in total). The online case is chosen, so that the measurement can focus on fine-grained change propagation; otherwise the majority of time would be spent with serializing and deserializing entire models and doing VCS checkouts/commits.

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.

5.1.2 Measurements

We measured[1] the change propagation on a personal computer[2] with maximum a 7GB of RAM. 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.

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 ranging from $U = 2$ to $U = 100$.

The results of the first series is shown by Figure 4. 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.

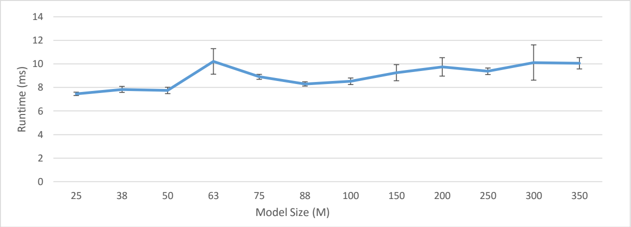

Figure 4: Average Execution Time of a Signal Reversal (varying model size)

The results of the second series is shown by Figure 5. 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.

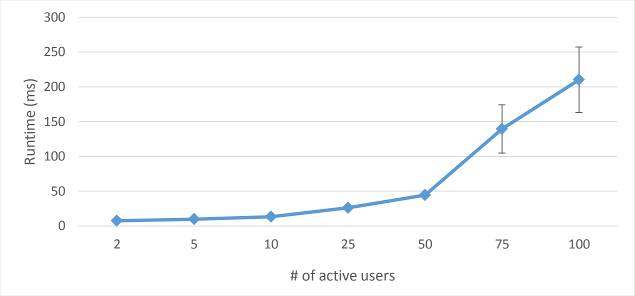

Figure 5: Average Execution Time of a Signal Reversal (varying # of concurrent users)

5.2 Assumptions and Limitations

5.2.1 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

[1] Raw data and reproduction instructions at http://tinyurl.com/models16-access-control

[2] CPU: Intel Core i7-4700MQ@2.40GHz, MEM: 8GB, OS: Windows 10

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.

5.2.2 Permanent Identifiers for Offline Collaboration

Before executing the PutBack transformation rules in the offline scenario, the newly submitted front model has to be loaded on the server, and the *object correspondence* (see Sec. 4.2) must be initialized. The challenge is identifying which elements in the front model correspond to which elements of the target model. One solution is to introduce some kind of permanent unique identifier for all model elements. [3] Such a permanent identifier is preserved across model revisions and lens mappings, and can therefore be used to pre-populate the object correspondence relation. Note that unlike EMF, some modeling platforms (e.g. IFC [25]) automatically provide such permanent identifiers.

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 (omitted from Figure 1).

5.2.3 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* [33], 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.

5.2.4 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 [13] 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.

6. RELATED WORK

Access control in RDF triple/quad stores. Graph-based access control is a popular strategy for many triple and quad stores (4store [22], 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 [20] 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 [12]; 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 [31], 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 [31] and are generalized into abstract tokens and operators in [32].

Context-dependent access control [2, 8] 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 [15].Post-filtering query results to enforce the access control policy is also possible [6] but this strategy may have performance issues without dedicated support.

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 [18] 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 [9], WebGME [29], or the tools developed according to [21].

AToMPM [35] 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 [14].

The VehicleFORGE collaborative hardware design platform has an access control scheme TrustForge [10] 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.

Model-driven security. Model-based techniques have also been used for access control purposes. In [26], 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 [7], which specifies access control policies by OCL. Although this

[3] 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.

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 [28] 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.

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, [17] 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 [30], 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.

Other. Graph or logic based, declarative specification formalisms (like Datalog programs or graph transformation) can be applied for validating access control policies [27, 5, 37]. 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 [23], 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.

7. 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 and increasing number of collaborators. Initial results for the online collaboration case were promising with close to instant propagation of changes and checking of write permissions up to 75 simultaneous collaborators.

As future work, we would like to primarily address the limitations presented in Sec. 5.2, and formally characterize the assumptions under which Get and PutBack terminate and satisfy other correctness criteria (incl. undoability) found in bidirectional transformations literature.

8. REFERENCES

[1] CAESAR Research Project. http://store.sae.org/caesar/.

[2] Fabian Abel, Juri Luca De Coi, Nicola Henze, Arne Wolf Koesling, Daniel Krause, and Daniel Olmedilla. Enabling advanced and context-dependent access control in RDF stores. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *LNCS*, pages 1–14. Springer, 2007.

[3] Aerospace vehicle systems institute. SAVI Research Project. http://http://savi.avsi.aero/.

[4] Apache. Subversion. https://subversion.apache.org/.

[5] Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. *ACM Transactions on Inf. and System Security*, 6:71–127, 2003.

[6] Joachim Biskup and Torben Weibert. Confidentiality policies for controlled query evaluation. In *Proc. of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, volume 4602 of *LNCS*, pages 1–13, Redondo Beach, California, USA, July 2007. Springer.

[7] Ruth Breu, Gerhard Popp, and Muhammad Alam. Model based development of access policies. *International Journal on Software Tools for Technology Transfer*, 9(5):457–470, 2007.

[8] Willy Chen and Heiner 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*, volume 246 of *books@ocg.at*, pages 663–672. Österreichische Computer Gesellschaft, 2009.

[9] Cauê Clasen, Frédéric Jouault, and Jordi Cabot. VirtualEMF: a model virtualization tool. In *Advances in Conceptual Modeling. Recent Developments and New Directions*, volume 6999 of *LNCS*, pages 332–335. Springer, 2011.

[10] Penn University DARPA VehicleFORGE. *TrustForge: Flexible Access Control for VehicleForge.mil Collaborative Environment*, 2012.

[11] Csaba Debreceni, Gábor Bergmann, István Ráth, and Dániel Varró. Deriving effective permissions for modeling artifacts from fine-grained access control rules. In *Int. Workshop on Collaborative Modelling in MDE (COMMitMDE)*, 2016. Submitted.

[12] Sebastian Dietzold and Sören Auer. 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.

[13] Zinovy Diskin. Algebraic models for bidirectional model synchronization. In *MoDELS*, pages 21–36, 2008.

[14] Matthias Farwick, Berthold Agreiter, Jules White, Simon Forster, Norbert Lanzanasto, and Ruth 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*, volume 6189 of *LNCS*, pages 278–291. Springer, 2010.

[15] Giorgos Flouris, Irini Fundulaki, Maria Michou, and Grigoris Antoniou. Controlling access to RDF graphs. In *Future Internet - FIS 2010*, volume 6369 of *LNCS*, pages 107–117. Springer, 2010.

[16] Karl Franz Fogel and Moshe Bar. *Open source development with CVS*. Coriolis Group Books, 2001.

[17] J. Nathan Foster, Benjamin C. Pierce, and Steve Zdancewic. Updatable security views. In *Proceedings of the 2009 22Nd IEEE Computer Security Foundations Symposium*, CSF '09, pages 60–74, Washington, DC, USA, 2009. IEEE Computer Society.

[18] The Eclipse Foundation. CDO. http://www.eclipse.org/cdo.

[19] The Eclipse Foundation. EMFStore. http://www.eclipse.org/emfstore.

[20] Inc. Franz. AllegroGraph. http://franz.com/agraph/allegrograph/doc/security.html.

[21] Jesús Gallardo, Crescencio Bravo, and Miguel A. Redondo. A model-driven development method for collaborative modeling tools. *J. Network and Computer Applications*, 35(3):1086–1105, 2012.

[22] Garlik. 4store. http://4store.org/trac/wiki/GraphAccessControl.

[23] Mikhail I. Gofman, Ruiqi Luo, Jian He, Yingbin Zhang, and Ping Yang. Incremental information flow analysis of role based access control. In *Proc. of the 2009 International Conference on Security and Management*, pages 397–403, Las Vegas, Nevada, USA, 2009. CSREA Press.

[24] David Hearnden, Michael Lawley, and Kerry 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*, volume 4199 of *LNCS*, pages 321–335, Genova, Italy, October 2006. Springer.

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

[26] Jan Jürjens. Model-based run-time checking of security permissions using guarded objects. In Martin Leucker, editor, *Proc. of the 8th International Workshop on Runtime Verification*, volume 5289 of *LNCS*, pages 36–50, Budapest, Hungary, 2008. Springer.

[27] Manuel Koch, Luigi V. Mancini, and Francesco Parisi-Presicce. A graph-based formalism for RBAC. *ACM Transactions on Information and System Security*, 5(3):71–127, August 2002.

[28] Levi Lucio, Qin Zhang, Phu Hong Nguyen, Moussa Amrani, Jacques Klein, Hans Vangheluwe, and Yves Le Traon. Advances in model-driven security. *Advances in Computers*, 93:103–152, 2014.

[29] Miklos Maroti, Tamas Kecskes, Robert Kereskenyi, Brian Broll, Peter Volgyesi, Laszlo Juracz, Tihamer Levendovszky, and Akos Ledeczi. Next Generation (Meta)Modeling: Web- and Cloud-based Collaborative Tool Infrastructure. In *8th Multi-Paradigm Modeling Workshop*, Valencia, Spain, 09/2014 2014.

[30] Lionel Montrieux and Zhenjiang Hu. Towards attribute-based authorisation for bidirectional programming. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*, SACMAT '15, pages 185–196, New York, NY, USA, 2015. ACM.

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

[32] Vassilis Papakonstantinou, Maria Michou, Irini Fundulaki, Giorgos Flouris, and Grigoris 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*, pages 103–112. ACM, 2012.

[33] Perdita Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *Software & Systems Modeling*, 9(1):7–20, 2008.

[34] Michael Stonebraker and Eugene Wong. Access control in a relational data base management system by query modification. In Roger C. Brown and Donald E. Glaze, editors, *ACM '74: Proceedings of the 1974 annual conference - Volume 1*, pages 180–186, New York, New York, USA, 1974. ACM.

[35] Eugene Syriani, Hans Vangheluwe, Raphael Mannadiar, Conner Hansen, Van Mierlo, and Huseyin Ergin. AToMPM: A Web-based Modeling Environment. MODELS 2013 Demonstrations Track, 2013.

[36] The Eclipse Project. Eclipse Modeling Framework. http://www.eclipse.org/emf/.

[37] Romuald Thion and Stéphane Coulondre. Representation and reasoning on role-based access control policies with conceptual graphs. In *Proc. of the 14th International Conference on Conceptual Structures*, volume 4068 of *LNCS*, pages 427–440, Aalborg, Denmark, 2006. Springer.

[38] Zoltán Ujhelyi, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, Benedek Izsó, István Ráth, Zoltan Szatmári, and Dániel Varró. EMF-IncQuery: An integrated development environment for live model queries. *Science of Computer Programming*, (0):–, 2014.

[39] Jon Whittle, John Hutchinson, and Mark Rouncefield. The state of practice in model-driven engineering. *IEEE Software*, 31(3):79 – 85, 2014.