ARTICLE IN PRESS

Science of Computer Programming ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

A model-integrated authoring environment for privacy policies

Andras Nadas [a], Tihamer Levendovszky [a,*], Ethan K. Jackson [b], Istvan Madari [a], Janos Sztipanovits [a]

[a] Vanderbilt University, Nashville, TN, United States

[b] Microsoft Research, Seattle, WA, United States

HIGHLIGHTS

- Model-Integrated Privacy Policy Authoring environment for lawyers and doctors.
- Precise semantic anchoring to term algebras and logic programs.
- A domain-specific language describing privacy policies with patterns.
- An execution environment by the tool FORMULA to prove correctness by design.
- A formally underpinned environment to reason about privacy policies.

ARTICLE INFO

Article history:
Received 8 March 2012
Received in revised form 2 May 2013
Accepted 5 May 2013
Available online xxxx

Keywords:
Privacy policies
Model-integrated computing
Constraint logic programming

ABSTRACT

Privacy policies are rules designed to ensure that individuals' health data are properly protected. Health Information Systems (HIS) are legally required to adhere to these policies. Since privacy policies are imposed on complex software systems, it is extremely hard to reason about their conformance and consistency. In order to address this problem, we have created a model-driven authoring environment to formally specify privacy policies originally defined in legal terms. In our observation, appropriate formalization of our policy language enabled formal analysis of its policies; these features were key to a successful model-driven engineering process. In this paper we present our modeling language and show its semantic anchoring to analyzable logic programs. We report on several projects where our approach is being applied and validated.

1. Introduction

Secure access to health information has become vital to cost-effective health care delivery. However, since the advent of electronic medical records and health information exchange infrastructures, the number of laws governing patient privacy has been growing every year. Naturally, the likelihood of inconsistent or contradictory policies is increasing with the number of regulatory agencies and offices. For example, in the United States health information systems must operate according to federal and state laws, and local institutional policies created by the privacy officer of various health care organizations. In summary, the growing regulatory environment presents a considerable challenge for the development of health information systems.

* Corresponding author. Tel.: +1 6156689827.
E-mail addresses: andras.nadas@vanderbilt.edu (A. Nadas), tihamer@isis.vanderbilt.edu (T. Levendovszky), ejacksonh@microsoft.com (E.K. Jackson), pityu@isis.vanderbilt.edu (I. Madari), janos.sztipanovits@vanderbilt.edu (J. Sztipanovits).

0167-6423/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.scico.2013.05.004

Clearly, methods for automatically enforcing policies are needed. Formally specified privacy policies have the advantage of automated enforcement and consistency checking. There are several conceptual formal frameworks for accomplishing this. One well known approach is the work of Nissenbaum and Mitchell on contextual integrity [1], which states that privacy requirements must be understood within the social contexts producing them. Their formalization must include this social context [2].

Following these ideas, we present a model-based policy authoring framework called *PATRN* (**P**olicy **A**u**t**horing and **R**easoni**n**g). Our contributions are:

- **Domain-specific language.** We present a domain-specific language (DSL) for precise policy authoring. It allows non-experts to easily write formal policies.
- **Formalization and analysis.** We formalize PATRN using the FORMULA specification language, which is based on logic programming. We show that this approach facilitates automated analysis of policies. We use a technique called semantic anchoring that connects the model-based environment with the formal specification.
- **Validation.** We report on several projects where PATRN is being applied and its efficacy validated. We describe the successes and challenges encountered thus far.

The paper is organized as follows: First, we present related work in Section 2. Section 3 describes the PATRN DSL that successfully enables the modeling of the wide variety of privacy policies found in Health Information Systems. Detailed information is provided on each of the components of the language including the ontologies, the templates, the semantic anchoring, and the policy models themselves. Section 4 reviews the mathematical background of the FORMULA specification language, and Section 5 demonstrates the formalization of PATRN with FORMULA. Section 6 illustrates automated analysis of PATRN policies using FORMULA. Section 7 reports on successes and challenges observed when applying PATRN, and we conclude with a discussion of future work in Section 8.

2. Background and related work

The Health Insurance Portability and Accountability Act (HIPAA) was introduced in 1996. Since its introduction there have been efforts [3] to formalize its federal-level privacy policies; these are usually referred to as "HIPAA policies". In 2010 the Health Information Technology for Economic and Clinical Health (HITECH) Act added more regulations and policies. Our work focuses on creating an infrastructure that helps formalize and analyze the increasing number of privacy policies for health information systems. The formalization of privacy policies enables the analysis, composition and transformation of the policy frameworks [4].

Many existing analysis approaches focus on system software and not the policies themselves. The work described in [5] analyzes interactive systems using probabilistic bisimulation relations. Our method is not probabilistic, but reasons about policy specification using a *finite model finder*. Most of the existing work regarding the formalization of privacy policies was accomplished to enable the enforcement of the policies in systems they govern [6,7]. It has been shown that formal privacy policies can be used to automate audit procedures based on logs that systems create at run-time [8,9]. The formalization is usually carried out with logic programming languages, typically PROLOG [10], or with temporal logic [11]. SecPAL [12] focuses on the creation of extensible authorization languages and their formal semantics that enables authorization in large distributed systems. It provides formal semantics and declarative predicates of principals that helps describe the domain of the authorization rules and maps them to Datalog for evaluation.

While these approaches aim at consistent manual or semi-automated encoding of policies in logic terms, mostly compliant with PROLOG specifications, our method introduces a separation of concerns. A domain-specific language with well-defined and extensible constructs precisely represents the key concepts (ontology) and allows reusable and user-defined templates. The formalisms of the logic terms of the other approaches can be expressed with our template language, however, our approach also supports their extensions, too. As opposed to policy analysis, only a few approaches go beyond audit or enforcement by providing formal analysis. [13] contributes a formal textual language to describe privacy policies. In contrast to our approach, they do not use graphical models, and their language does not include extensibility mechanisms. However, the semantics of the language is also a logic program extended with formal data types. This semantics, however is not customizable as our template-based method.

Contrary to formalisms that are written in a logic programming language [10], the formal models presented in this paper strictly enforces the syntax of the building blocks and provide formal models that are syntactically correct by construction. The extensibility of our approach coupled with the ontology models can also formalize the context and domain of the policies to a greater extent. Such formal model elements enable finer refinement of the policy rules by the formalization of hierarchies and relations of certain objects and entities, consequently fulfilling a further requirement for adoption in real systems as described in [10].

The semantics of domain-specific languages has been the focus of many research efforts [14–16], including semantics for the purpose of verification. The formal semantics of general modeling languages, such as UML [17–19], and programming languages [20–23] has also been extensively studied.

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

Table 1
Examples of the policies that were formalized using PATRN, including a brief explanation.

Level	Name	Content
Federal	HIPAA Opt-out	Patient can opt out from all disclosures made through an HIE.
State	Disclosures of General Health Information	In most states General Health Information can be disclosed for treatment without Patient consent.
State	Disclosures of Medication History	In some states Medication History can only be disclosed to a treating Physician without Patient consent.
State	Disclosures of Mental Health Reports	In most states Mental Health Reports can only be disclosed with written consent or to Physicians that already treat the patient
State	Disclosures of HIV Reports	In some states HIV reports can only be disclosed in case of Emergency or with written consent from the Patient
Institution	Disclosures of Protected Health Information (PHI) to third parties	Disclosure of PHI can only happen with Patient consent or to Insurance companies for payment or to an External Provider for treatment
Institution	Use of Protected Health Information (PHI) with IRB approval	PHI can be only be used for treatment unless there is an IRB approval for non-treatment use.
Institution	Access to Protected Patient Information (PPI) by Job Role	PPI can only be accessed by employees with certain Job Roles, mainly Clinicians but also Administrators for Payment processing.

Previous work has shown that domain-specific modeling languages are more reusable with the help of semantic anchoring [24]. Semantic anchoring for domain specific languages has been accomplished with Abstract State Machines [25–27] and logic programming languages [28].

Using modeling and semantic anchoring, it is possible to extend and incorporate already existing research results into the formal policy models. More specifically, the template models that are presented in detail in Section 3.4 enables the incorporation of established privacy languages such as the ones presented in [13] into the modeling environment. Furthermore, it is possible to leverage the established ontologies [29] to be used as the domain language in parallel to the other privacy languages. It is also possible to integrate privacy languages as the language of the semantic anchors. In this case the semantics of the templates would be written using the formalisms established by the privacy language, thus providing all the analysis capabilities of the privacy language to the formal policy models.

FORMULA (Formal Modeling Using Logic Programming and Analysis) is a formal modeling language based on Algebraic Data Types (ADTs) and Constraint Logic Programming (CLP). FORMULA has been shown to be able to formalize structural semantics of domain-specific modeling languages [30]. It has also been proven that FORMULA can reason about how non-functional requirements impact the system and platform design space [31]. The capabilities of FORMULA to be able to reason on domain-specific languages with metamodeling frameworks were also demonstrated [32].

3. PATRN: the MDE tool for formal policy models

Our initial goal was to greatly expand the set of formalized policies by including state policies, institutional policies and HITECH policies besides the already formalized HIPAA policies. When we reviewed these policies, we realized that the many of the policies follow similar patterns. For example, the policies that control the disclosure of the medication history of patients are very similar in the states California and Tennessee. While the policy of California is stricter, both policies relate the same concepts in similar context. The main difference between the policies is in the constraints that they impose on the patient and physician. By creating templates that can cover the policies that share the same characteristics, it becomes possible to simplify the formalization process. Table 1 shows a brief list of the policies we examined and later modeled with PATRN. The listed state level policies regulating the disclosures of highly sensitive information were modeled for several states including California, Tennessee and New Hampshire.

We also found that the policies use objects and entities that can be classified and organized using ontologies. Combining these observations allowed us to create a formalization framework for privacy policies. The four pillars that underpin the policy formalization are as follows: (1) reusable constraint patterns, (2) ontologies for the entities, (3) a logic programming language for the formal description, and (4) an execution environment for the automated reasoning. Minor parts of the code generators have been written in a general purpose programming language. The framework is shown in Fig. 1.

The PATRN policy framework makes the formalized policies highly reusable in many different domains. With anchoring the semantics for analysis, it is possible to detect conflicts, implications, entailment and equivalence between policies or policy sets. It is also possible to anchor operational semantics to the patterns and compose and export the formalized policies into an execution framework that enforces them in a health information system using the defined semantics.

Our cooperation with medical institutions has shown that this framework is able to bridge the gap between the policy makers, privacy officials, and health information system engineers by providing a common, formal, and unambiguous system of specification and verification.

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

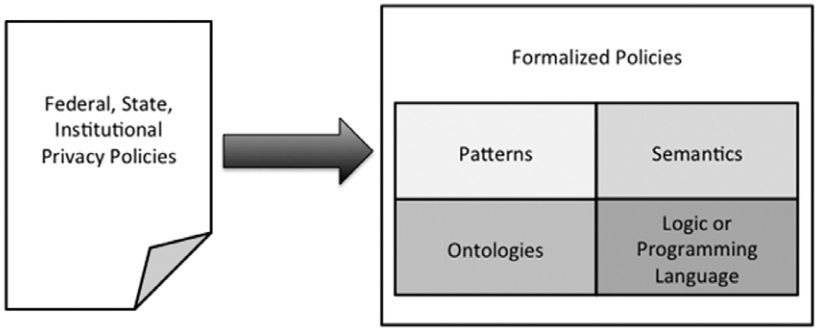

Fig. 1. The overview of the components of the policy formalization framework.

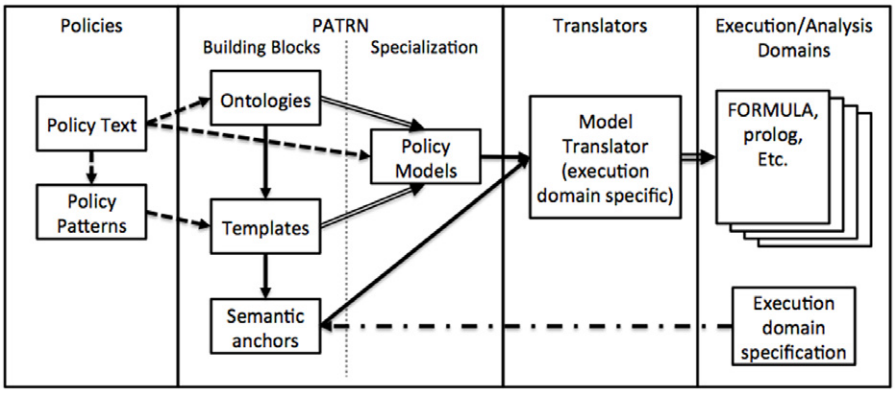

Fig. 2. The architecture of the PATRN environment.

3.1. PATRN introduction

The PATRN toolkit is a Model-Integrated Computing (MIC) toolkit comprised of several MIC tools to provide an integrated environment to formalize, compose, and reason about privacy policies. The environment consists of the following main components. (i) A domain-specific modeling environment implemented in GME [33]. This describes the possible entities and their relationships with ontologies, as well as the policies defined with a set of reusable hierarchical constraints over the entities. (ii) Textual implementations of the constraints on the lowest level of the hierarchy. The formal policy models are composed together from the ontologies and policy templates.

The formalization of the policies with the PATRN toolkit is achieved as shown in Fig. 2. The first step is to recognize the common patterns used in the textual policy descriptions. These patterns will form the templates in PATRN. The next step is to compile the object and actors of the policies and organize them into ontologies. Once the templates and ontologies are defined, formal policy models can be composed from them.

Domain-specific languages formalize the syntax of the well-formed models only by means of metamodeling. In order to utilize formal machinery, we need to assign formal semantics to our domain-specific models. In fact, the semantics of the templates are assigned directly by the modeler at design time by providing logic program fragments. These fragments are completed automatically when the templates are specialized as shown in Section 5.2. The formal policy models receive their semantics from the templates which they are composed of. Eventually, templates are anchored to semantic templates. When the patterns are specialized, the semantic templates are automatically specialized. Via this mechanism, we can automatically assign specialized semantics to the policies themselves. Once the semantic anchors are provided to the formal policy models through the templates, the formal policy models can be interpreted and translated into the language of the execution environment for analysis or enforcement. We emphasize that we do not focus on the problem of policy translation from natural languages to our models.

In the following sections, we present the four components: the ontologies, the templates, the formal policy models and the semantic anchoring mechanisms as well as how they relate to each other. We also present a use case based on real world policies and demonstrate how the use case is formalized using the four components. Our target domain is a logic programming tool that enables us to analyze and compare the formal policy models to detect contradictions, entailment and equivalence. The logic programming tool used is FORMULA and the semantic anchoring is provided in a syntax close to that of FORMULA. The translator that we use is a simple interpreter in GME that exports the ontologies into FORMULA, using FORMULA's own modeling language, and then expands the semantic anchors into FORMULA's logical expression language.

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

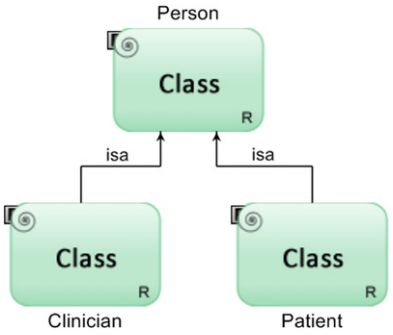

Fig. 3. A simple model describing a taxonomy of persons.

3.2. Use case: state policies for releasing medication history

The use case that will show the functionality of our framework is taken from a comprehensive report [34] on how different states handle the privacy aspect of exchanging Health Information. The report provides a unified and harmonized view on how the disclosure of sensitive information is governed by the laws in each state. This harmonized and unified view enables us to create use cases without the confusion stemming from the verbal differences in the laws themselves.

As an illustrative case study, we examine the disclosure of pharmacy record as sensitive health information. The requirements for this vary from state to state, this policy provides us a simple but rich set for experimentation. While the Tennessee laws and regulations do not limit the disclosure of pharmacy records, those in California limit the disclosure to parties that can provide proof that they are in a treatment relation with the patient.

As for the Tennessee law:

> **Clinicians can access pharmacy records on patients without any restriction.**

As for the California law:

> **Clinicians can access pharmacy records on patients only with proof that clinician and the patient are in a treatment relation.**

Even after a verbiage is given to the policies, they are still treated to be unified and harmonized. This means that the same words must mean the same in both policies.

3.3. Ontologies

The component that describes the relationship between entities in PATRN are ontologies. All of the other components, the templates and the formal policy models, use the ontologies directly or model elements reference them. The ontologies provide the facility to formally capture entities and their relationships that are referenced and used in the policies. The ontology language in PATRN is a graphical derivative of the Ontology Web Language (OWL) [29] that is specified by the metamodel of PATRN. The graphical language is capable of describing classes, individuals, inheritance relationships, and properties with the same semantics as specified by OWL.

The reason why the semantics of OWL was adopted to the PATRN toolkit was to enable the integration, import and export of ontologies from already existing sources and systems. Any ontology created in PATRN can be exported into a standard RDF XML [35] file. Fig. 3 shows a small example of a person taxonomy describing that a clinician and a patient are both persons and it is modeled after the use case in Section 3.2. To show how the PATRN ontology language matches OWL, the same taxonomy is shown as an RDF in Fig. 4.

In PATRN, classes are used to classify and declare elements that are used in policy templates and policies. PATRN provides means to specify inheritance or 'is a' relations between Classes. Specifying the inheritance relationship is possible by assigning either sub-classes or super-classes to a class. While they are equivalent in expressiveness, supporting both kinds of inheritance specification method gives the users a choice to describe their ontologies in a way that better suits their domain. In the PATRN language, classes are represented by GME models. These models can contain two different kinds of references to other classes, one to specify sub-classes and one for super-classes as shown in Fig. 5.

PATRN also supports multiple inheritance between classes with the semantics provided by OWL. Multiple inheritance enables the users to create multiple ontologies that are interconnected by common shared elements. With the multiple ontologies, users can add new ontologies to the system and making certain that it will fit the already existing ones.

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
  <rdf:Description rdf:about="http://isis.vanderbilt.edu/sharps/PATRN/PersonOnt/Patient">
    <rdfs:subClassOf rdf:resource="http://isis.vanderbilt.edu/sharps/PATRN/PersonOnt/Person"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://isis.vanderbilt.edu/sharps/PATRN/PersonOnt/Person">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://isis.vanderbilt.edu/sharps/PATRN/PersonOnt/Clinician">
    <rdfs:subClassOf rdf:resource="http://isis.vanderbilt.edu/sharps/PATRN/PersonOnt/Person"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  </rdf:Description>
</rdf:RDF>
```

Fig. 4. RDF description of the person taxonomy in Fig. 3.

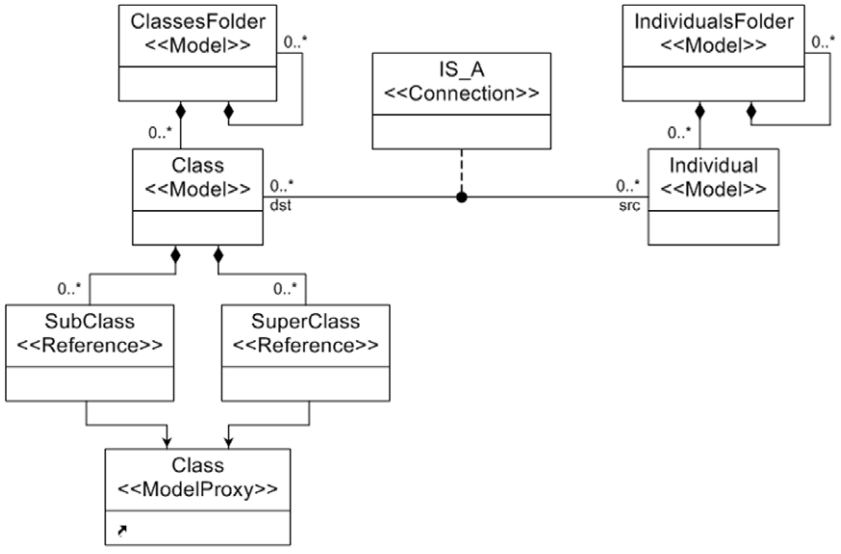

Fig. 5. The ontology language described in the GME metamodel language.

Another very important type of model in PATRN is the model of individuals. Individuals represent instances that are members of classes defined; shown on the right side of Fig. 5. The terminology of Individual was adopted from OWL where it represents Class instances. These models of individuals are used in formal policy models to describe actors and objects in the formal policy models. In the formal policy models individuals are used to model either a certain person such as Dr. Smith, or persons in certain role or function such as 'the treating physician'. With individuals it is not only possible to model Persons but also instances of objects like 'the medical record of John Doe'.

3.4. Policy patterns: the policy templates

In the center of the PATRN toolkit is a graphical language that describes the *templates* for the formal policy models. Templates can represent patterns of relations between individuals, constraints and most of all policy language patterns. PATRN supports the creation of any number of templates for any usage scenario. They have a special purpose when viewed from the point of metamodeling. The templates and the ontology elements together describe a modeling language. It is a sub-language of the language specified by the PATRN metamodel. When viewed from this angle it is quite clear that the template creation is a metamodeling activity and such requires in depth domain knowledge from the template authors.

The template authoring and curation is an iterative process. It is similar to the creation of patterns for Object Oriented (OO) programming, the difference is the starting point. In the case of the OO programming the patterns are established based on programming best practices, but for policy templates the templates are established based on the understanding of how the policy text are written and not from already existing policy models. Once a set of templates has been established, the templates can be iteratively refined, extended and curated until the new set of templates cover the target domain.

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

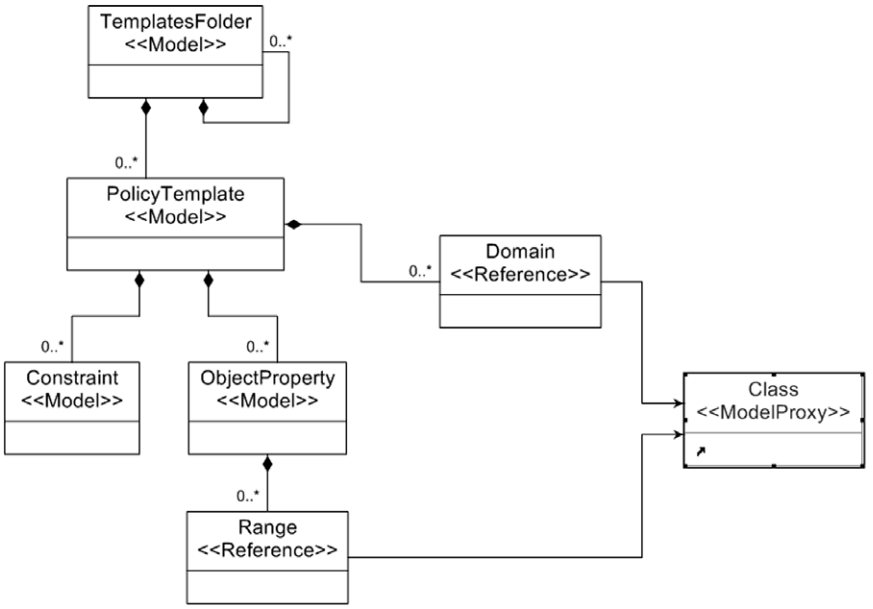

Fig. 6. The template language described in the GME meta language.

The templates provide a reusable structure with structural semantics enforced upon specialization of the pattern. The templates themselves do not imply or possess any denotational semantics. However, these can be given to the templates by anchoring their semantics with a formal specification that matches their intended final use. This separation of the structural and denotational semantics enables the use of the same patterns and their instantiated models in different target domains, such us analysis, verification and/or execution. Templates can be viewed as an extension of the OWL properties, and represent an *n*-ary association between individuals of classes, where the different roles of the association can have different properties assigned to them.

Each template is built from properties, constraints and a single relation domain as shown in Fig. 6. The properties and constraints represent the facets of the templates that are filled at specialization time. The properties in PATRN are analogous to the properties in OWL. They specify an association to the other parts of the specialized pattern. The domain of the template specifies the class of the template that is taken from the ontology. In most cases the templates will have a domain that is a policy template class. Templates can also describe other relationships such as a treatment relationship between a patient and a clinician. In this case the domain of the template will be a treatment relationship class.

The template language in PATRN is a simplified and more descriptive graphical language adaptation of the object properties of OWL. The result of this is that every instantiated template in PATRN can unambiguously be translated into OWL. Every pattern instance translated into OWL will have an individual as a center element that is a member of the class that is specified as the domain of the template. The individuals that are filling the facets of the template will be associated to the center individual with the object property specified for the facet by the template model.

In order to show how templates can be used to model policies and other relationships, two examples that cover the use case in Section 3.2 are described below. The first example, depicted in Fig. 7, illustrates a template for state policies. The domain of the template is specified as *Policy*. This means that the template is for describing policies in contrast to describing associations. The template is created to be able to describe the two policies of the use case—one used in California, one in Tennessee. To create the template, generalized descriptions of the policies were captured. This generalization is as follows.

Requestor who is a Person can access an Object that is a Medical Document of a Subject who is a Person only if the specified Constraint is satisfied.

The state policy template, that can describe all the variations of state policies from [34] for the disclosure of sensitive health information, is modeled to have four facets from which one is constraint specification. The other three facets are for specifying the actors and object of the template. The two actors of the policy are the *Requestor* and the *Subject*. Both have range specifications that require them to be filled at instantiation time by members of the *Person* class. The explicit declaration of the *Requestor* and *Subject* makes it possible to use these two actors in the constraint specification. The object of the template is a member of the *Medical Document* class.

Templates can also be used to model associations that have two or more participating individuals. Typically these associations describe relations or facts about actors and objects related to the policies. The templates can also be used to model logical expressions, with the freedom of using any logic or declarative programming language anchored as the

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

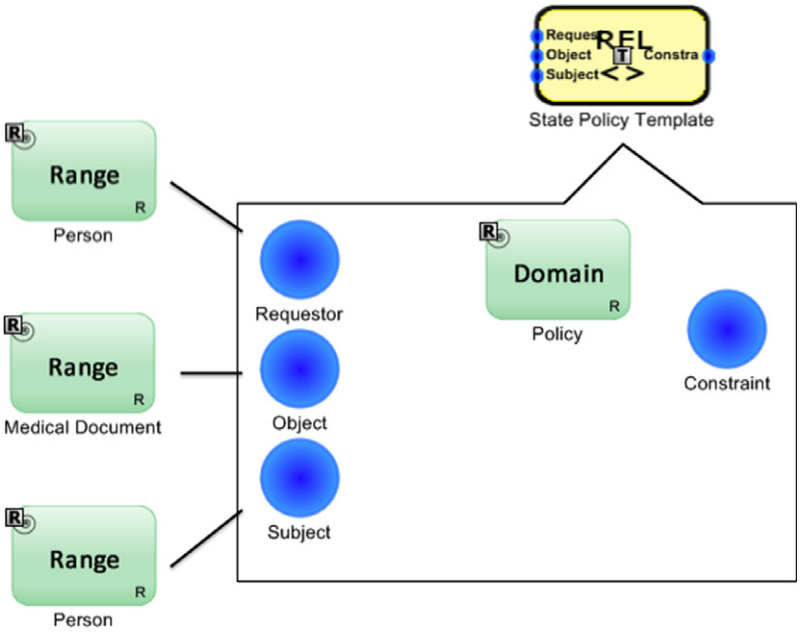

Fig. 7. The template for state policies described in the PATRN template language.

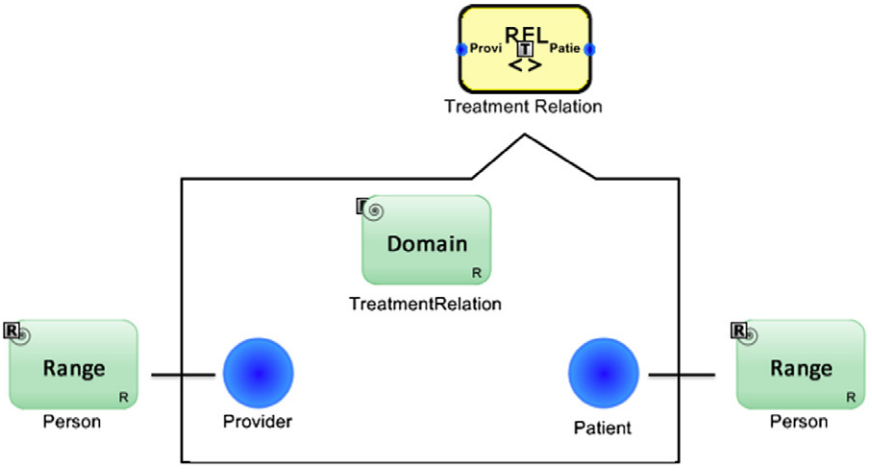

Fig. 8. The template for treatment relations described in the PATRN template language.

semantics of the template. Composing the ontologies and the templates that model relations and logical expressions enables a wide range of constraints to be precisely described.

An example of a template describing a relation between actors is the template for describing a treatment relation between a *Provider* and a *Patient*. The *Domain* of the template is specified as *Treatment Relation* and the two facets of the template are from the *Person* class as show in Fig. 8.

An example for describing a logic expression can be as simple as an OR expression describing the function X OR Y as shown in Fig. 9. The template for the OR expression will have a domain of Logic Expression and it would have two facets of X and Y that will be the sub constraints. The facets X and Y do not have Ranges associated to them as they are facets for Constraints similarly to the Constraints in the Policy Template. It is also not necessary to strongly type Constraints at this level of abstraction. The typing of the facets happen when the semantics are anchored to the template. Logic expressions are necessary for describing complex constraints such as the one in the policy:

Clinicians can access pharmacy records on patients only with proof that the clinician and the patient are in a treatment relationship OR the patient gave written consent to disclose pharmacy documents.

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

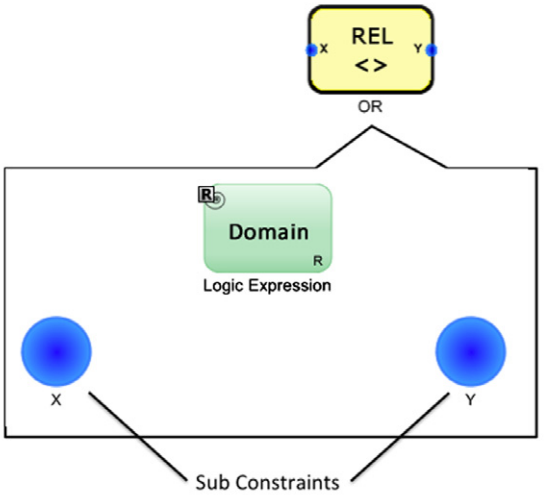

Fig. 9. The template for the logical expression described in the PATRN template language.

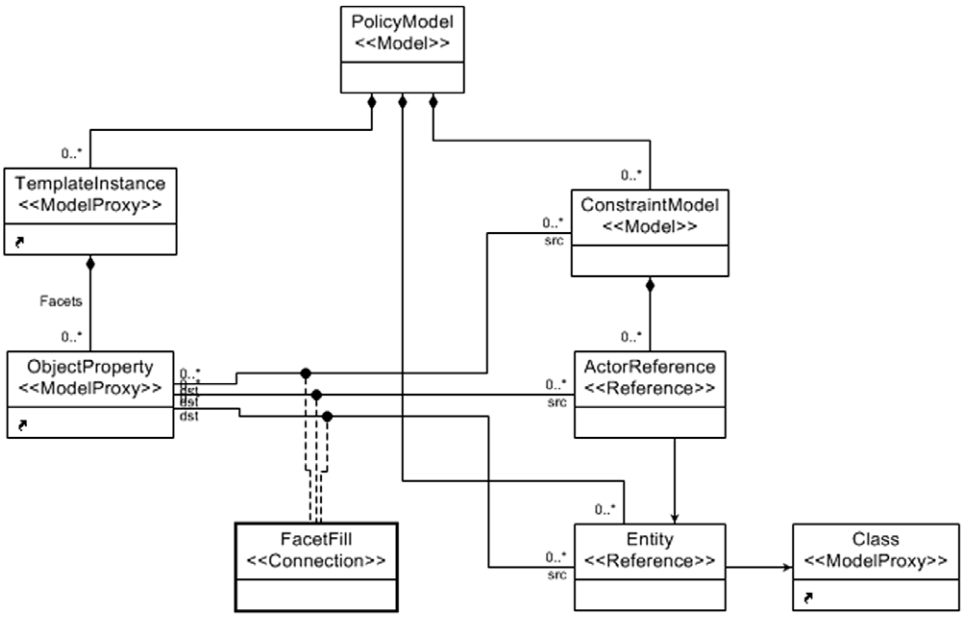

Fig. 10. The PolicyModel described in the GME meta language.

3.5. Template specialization

The formal policy models are the main objects of the PATRN toolkit. The representations of the formal policy models in the PATRN language are the *PolicyModels*. The *PolicyModels* are containers that encompass all the compositional parts of the formal policy models. The *PolicyModels* are built by composing the templates for the policies and associations with individuals derived from the ontologies as shown in Fig. 10.

To compose the templates with the individuals, the templates have to be specialized. The specialization is done by instantiation using the type inheritance mechanism [36] built into GME. The type inheritance framework enables the creation of type models and type instance models that are linked to the type models. In the case of the PATRN toolkit the type models are the templates, the policy and association models are the instances. The type instance infrastructure keeps the type models and the type instance models synchronized by applying asymmetric synchronization rules. All the changes made to the type models are propagated to its instance models, while no changes are allowed in the instance models that would break the type-instance relationship.

In *PolicyModels*, the instantiated templates are composed with the individuals that represent the actors and objects in the policy. The individuals are entities that are instances or members of a class of the ontologies as described in Section 3.3. The individuals for the two policies from Section 3.2 are the same, the two policies only differ in the constraint. The actors are

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

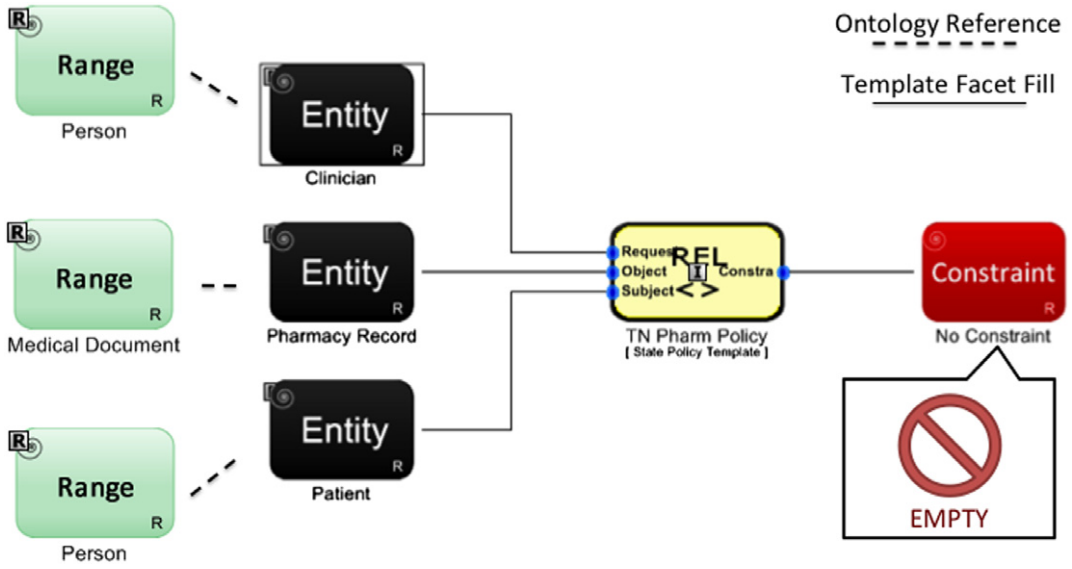

Fig. 11. The Tennessee Pharmacy Record disclosure policy described in the PATRN template language.

the *Patient* and the *Clinician* both of the policies. The object of the policies is the *Pharmacy Record*. These actors and object are connected to the facets of the instances of the policy template. The *Clinician* actor will fill the facet for the Requestor, the *Patient* for the *Subject* and the *Pharmacy Record* for the *Object*; as shown in Fig. 11. In case of the Tennessee pharmacy record policy, as there is no constraint on the disclosure, the constraint is modeled with an empty model.

The constraints of the policy models are modeled very similar to the policy models themselves. The constraints are built by using template instances, individuals derived from the ontologies and references of actors and objects. The templates that are most often used are templates for associations and logic expressions. These templates are instantiated the same way as the top level policy templates. The individuals are also created the same way as the actors and objects of the policy models. The only model type that is not used by the top level policy models are the references of actors and objects. These references enable the same actors and objects to be used to fill more than one facet of a template or fill facets in multiple templates.

In order to model the California pharmacy record policy, the template infrastructure is used to specify the constraint between the actor *Clinician* in the *Requestor* role and the *Patient* actor in the *Subject* role. The constraint established by the policy restricts the disclosure of the information only to cases where there is a treatment relation between the *Clinician* and the *Patient*. This is modeled by creating an instance of the *Treatment Relation* template under the constraint model filling in for the constraint facet of the *Policy Template* instance. The *Treatment Relation* pattern has two facets, one for the *Provider* actor and one for the *Patient* actor. To create a properly modeled constraint the actors that are used to fill in these facets cannot be different from the individuals of the top level policy model. Two actor references are used to fill in these facets, one referencing the *Patient* and the other one the *Clinician* as shown in Fig. 12.

3.6. Anchoring denotational or operational semantics to the templates

The template models of PATRN specify the abstract syntax of the policies and constraints modeled as described in Section 3.4. To provide the denotational semantics of the templates, they have to be specified and anchored to the template models. PATRN does not have a restriction on the language used to specify the semantics, but the environment used to execute or analyze has to understand the language used. To specify the semantics of a template, a semantic description must be associated to each template model. To provide denotational semantics to the templates in FORMULA a simple model that contains fragment expressions that generates FORMULA code is used as elaborated in 5.2.

3.7. Lessons learnt from modeling real-life policies

Throughout this section we provided an in depth description on how policies can be formally modeled using the PATRN toolkit. To successfully model a wide variety the domain with its policies must be understood first. The policies themselves are usually created and written by the policy makers on a very high abstraction level. Mapping down from an abstract level to formal one with concrete representation always have its caveats, such us gaps, conflicts and incompatibles. To minimize the impact of such caveats the formal models can be analyzed and curated by the domain experts.

We have found that well-established templates as primitives can be the basis of a quite extensive set of policies. Although these templates are created manually by domain experts, the code generation and the evaluation of the models composed

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

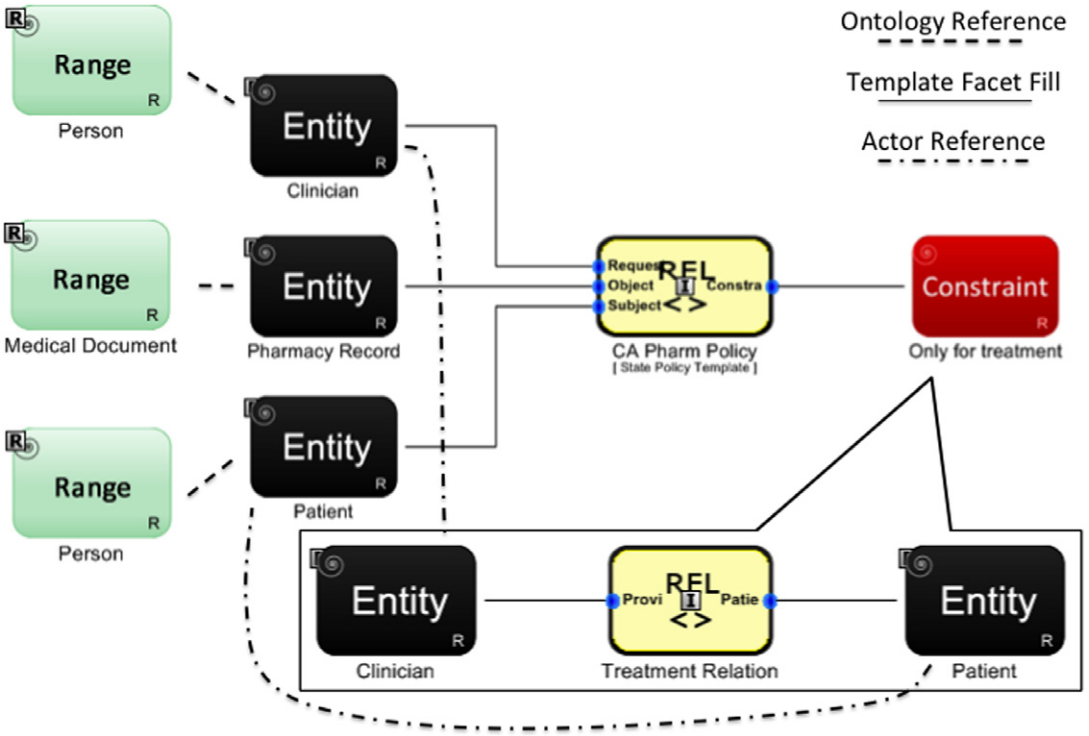

Fig. 12. The California Pharmacy Record disclosure policy described in the PATRN template language.

from the templates is automatic. The composition the primitives, the templates and the ontologies is currently performed manually from raw policy texts. The currently available policy texts are intended to be read by humans who are also domain experts. While there are publications that extract and analyze state policies [34] and could be used as a source for automatic model generation such sources are not generally available for the institutional and other policies.

We do think that in the future processing the policy texts with natural language processing (NLP) could provide a great tool to the policy modelers. The current developments in NLP for automatically deducting information from clinical texts and narratives [37] will be adaptable for policy texts. One of the missing pieces is a standardized terminology for the policy text, similar to the unified medical language system (UMLS) [38] in the medical domain. Also, the templates created for the policy modeling can be used as to provide the structure and context patterns for the NLP based extraction of the formal policy models from the policy texts.

4. Introduction to CLP and open world reasoning

Constraint logic programming (CLP) provides a powerful approach to writing formal specifications. First, a logic program Π can be directly (i.e. in polynomial-time) translated into *first-order logic* (FOL) according to its *Clark Completion*. Following the notation of [39], we refer to this translation as $\Pi^{\star}$. Second, logic programs are executable, allowing programmatic reasoning to be applied while devising specifications. This form of reasoning is harder to obtain when directly writing FOL. Consider the following program, which computes documents accessible within an organization.

Example 1 (*Accessibility*)**.**

$$\Pi_{access} \stackrel{def}{=} \begin{array}{lll} \mathsf{isAcc(id,\ x)} & \mathsf{:-} & \mathsf{doc(id,\ x).} \\ \mathsf{isAcc(id,\ x)} & \mathsf{:-} & \mathsf{isAcc(id,\ y),\ subOrg(y,\ x).} \\ \mathsf{isAcc(id,\ x)} & \mathsf{:-} & \mathsf{isAcc(id,\ y),\ subOrg(x,\ y).} \end{array}$$

The symbols $doc(,)$, $subOrg(,)$, and $isAcc(,)$ are user-defined data constructors. Applying data constructors to values constructs data. For instance, $doc(1, \text{“}PhA\text{”})$ is the application of $doc(,)$ to the constants 1 and “*PhA*”. It constructs a new value representing a medical document with ID 1 owned by a pharmacy named “*PhA*”. We call such applications *data terms*, or *terms* for short. A term is either a constant or an application of an n-ary constructor to n terms.

A FORMULA program corresponds to axioms about a set of terms K, called the *knowledge* of the program. Elements of K are called *facts*. There are two ways to influence the knowledge. One way is to directly place facts in the logic program:

$$\Pi_{TN} \stackrel{def}{=} \begin{array}{ll} \mathsf{doc(1,\ “PhA”).} & \mathsf{subOrg(“PhA”,\ “TN”).} \\ \mathsf{doc(2,\ “PhB”).} & \mathsf{subOrg(“PhB”,\ “TN”).} \end{array}$$

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

The program Π_{TN} contains four facts describing a few documents. When viewed as axioms about K, the program states that terms $doc(1, \text{“}PhA\text{”})$, $subOrg(\text{“}PhA\text{”}, \text{“}TN\text{”})$, $doc(2, \text{“}PhB\text{”})$, and $subOrg(\text{“}PhB\text{”}, \text{“}TN\text{”})$ elements of K.

The second way to influence the knowledge is through *rules*. For example, the program Π_{access} contains three rules. Each rule behaves similarly to a universally quantified implication, where the left-hand side (LHS) of the operator ':-' corresponds the head of the implication and the right-hand side (RHS) corresponds to the body. Whenever there is a substitution of variables for terms such that all terms of the RHS are facts, then the LHS must also be a fact for that same substitution.

4.1. Logical and execution semantics

Logic programs are more than just implications, because the only facts allowed in K are those explicitly stated in the program (as in Π_{TN}) or those that are forced to exist by rules. Formally, this means: (1) $\Pi^\star$ contains additional formulas to constrain the implications, and (2) the *intended interpretation* of $\Pi^\star$ is the least set K satisfying $\Pi^\star$ where every fact is justified by rules. For example, the Clark Completion for Π_{access} is:

$$\Pi^\star_{access} \stackrel{def}{=}$$
$$\forall id, x.\ isAcc(id, x) \in K \Leftrightarrow \begin{cases} doc(id, x) \in K \vee \\ (\exists y.\ isAcc(id, y) \in K \wedge subOrg(y, x) \in K) \vee \\ (\exists y.\ isAcc(id, y) \in K \wedge subOrg(x, y) \in K). \end{cases}$$

The least set K satisfying $\Pi^\star$ and justifiable by rules is called the *least model* of the program, written $lm(\Pi^\star)$. The class of logic programs supported by FORMULA guarantees the existence of least models. Therefore the least knowledge sets corresponding to the previous programs are:

$$lm(\Pi^\star_{access}) = \{\}.$$
$$lm(\Pi^\star_{TN}) = \begin{Bmatrix} doc(1, \text{“}PhA\text{”}), subOrg(\text{“}PhA\text{”}, \text{“}TN\text{”}), \\ doc(2, \text{“}PhB\text{”}), subOrg(\text{“}PhB\text{”}, \text{“}TN\text{”}) \end{Bmatrix}.$$
$$lm((\Pi_{access} \cup \Pi_{TN})^\star) = \begin{Bmatrix} isAcc(1, \text{“}PhA\text{”}), isAcc(2, \text{“}PhA\text{”}), \\ isAcc(1, \text{“}PhB\text{”}), isAcc(2, \text{“}PhB\text{”}), \\ isAcc(1, \text{“}TN\text{”}), isAcc(2, \text{“}TN\text{”}), \\ doc(1, \text{“}PhA\text{”}), subOrg(\text{“}PhA\text{”}, \text{“}TN\text{”}), \\ doc(2, \text{“}PhB\text{”}), subOrg(\text{“}PhB\text{”}, \text{“}TN\text{”}) \end{Bmatrix}.$$

Though Π_{access} contains rules for computing transitive accessibility of documents, it does not contain any documents. Therefore, the least model is empty. Π_{TN} contains only facts, but no axioms about accessibility, so its least model contains no additional terms. Finally, the combination of the programs produces a more interesting model containing the transitive accessibility of pharmacy documents in the state of Tennessee. (An alternative formalization for CLP is obtained by extending FOL with fixpoint operators [40].)

The primary operation on logic programs is the *query* operation. Let $t[\vec{x}]$ be a data term, but with variables $\vec{x} \stackrel{def}{=} x_1, \ldots, x_n$ appearing within the term. The operation $query(\Pi, t[\vec{x}])$ returns all the facts in the least model of Π that are of the form $t[\vec{x}]$:

$$query(\Pi, t[\vec{x}]) \stackrel{def}{=} \{t[\vec{x}\backslash\vec{s}] \mid t[\vec{x}\backslash\vec{s}] \in lm(\Pi^\star)\}.$$

The term $t[\vec{x}\backslash\vec{s}]$ is the term formed by replacing every variable x_i by the data term s_i in t. For example:

$$query(\Pi_{access} \cup \Pi_{TN}, isAcc(x, x)) = \{\}.$$
$$query(\Pi_{access} \cup \Pi_{TN}, isAcc(x, \text{“}TN\text{”})) = \begin{Bmatrix} isAcc(1, \text{“}TN\text{”}), \\ isAcc(2, \text{“}TN\text{”}) \end{Bmatrix}.$$
$$query(\Pi_{access} \cup \Pi_{TN}, isAcc(1, x)) = \begin{Bmatrix} isAcc(1, \text{“}PhA\text{”}), \\ isAcc(1, \text{“}PhB\text{”}), \\ isAcc(1, \text{“}TN\text{”}) \end{Bmatrix}.$$

Logic programs are also programs; they can be executed. Program execution is the application of rules to facts to produce more facts. A program terminates when it reaches a fixpoint where all facts have been produced. Typically, execution is triggered by a query operation, in which case the program may terminate as soon as the query is found to be satisfied for some term. Thus, reasoning about the least model does not require translating the program into FOL. Instead it is sufficient to execute the program.

4.2. Modeling and open world reasoning

FORMULA is logic programming tailored to formal modeling. It supports different types of specifications for formalizing abstractions and defining instances of these abstractions. FORMULA *domains* contain data type declarations and LP rules

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

axiomatizing abstractions. For example, the program Π_{access} would correspond to a domain axiomatizing document accessibility across organizations. Unlike in other LP languages, domains typically do not contain facts, i.e. their least models may be empty. This does not mean domains are useless. Instead, they are viewed as *open world specifications*. Later they will be exposed to new facts and they will compute useful information.

An instance of an abstraction, such as a particular database of documents, is called a *model*. (This is a different use of the word "model" than its earlier use in "least model".) A model is a logic program that only contains facts, like the Π_{TN} program. The semantics of a model is obtained by combining it with a domain, as in $\Pi_{access} \cup \Pi_{TN}$. Once a model is combined with its domain, then queries can be evaluated: "Is document *id* accessible by *x*?", "Can *x* access any documents?". The dichotomy of domains and models separates the fundamental ingredients of an abstraction from particular instances.

FORMULA carries this metaphor even further allowing new types of reasoning on domains:
Given domain D does there exist a model M where $query(t[\vec{x}], D \cup M)$ is non-empty?
This is a harder question to answer than standard query evaluation. We call this the *open world* query operation:

Definition 1 (*Open World Query Operation*)**.** Given:

1. A program Π with data constructors $\Sigma_\Delta \stackrel{def}{=} \{f_1, f_2, \ldots, f_n\}$,
2. $\Sigma_p \subseteq \Sigma_\Delta$ a subset of the constructors, called the *primitive* constructors.
3. A term $t[\vec{x}]$.

Then find a finite logic program M containing only facts where every fact uses only primitive constructors Σ_p and:

$$query(\Pi \cup M, t[\vec{x}]) \neq \emptyset.$$

We write $query_{owa}(\Pi, \Sigma_p, t[\vec{x}])$ for the open-world version of the query operation.

For instance, to find a database witnessing the *isAcc* property we may ask $query_{owa}(\Pi_{access}, \{doc, subOrg\}, isAcc(id, x))$. Notice that only $\{doc, subOrg\}$ are primitive constructors. This constrains the search problem so that an *isAcc* fact cannot be directly added to M. Instead, only *doc* and *subOrg* facts can be added to M, which in turn must cause the production of an *isAcc* fact. An example of a model returned by this query is:

$$\Pi_{sol} \stackrel{def}{=} \text{doc(0, "A"). subOrg("A", "B").}$$

FORMULA solves an open world query by constructing the satisfying model M. Because there may be infinitely many possible models to consider, search is implemented by efficient forward *symbolic execution* of logic programs into the state-of-the-art *satisfiability modulo theories* (SMT) solver *Z3* [41]. As a result, specifications can include arithmetic, data types, and variables ranging over infinite domains. Nonetheless, the method is constructive; it returns extensions of the program witnessing satisfaction.

4.3. Applications to privacy policies

Logic programs and open-world reasoning fit naturally with privacy policies. Policies can be succinctly expressed as logic programs [42,12]. Verification that the policy prevents anomalous access to sensitive information is equivalent to an open-world query for which there is no solution. If a solution is found, then it serves as a counterexample showing the system state where anomalous behavior is possible. Open-world queries can also show the relationships between policies: "Is there a world where policy P_1 is satisfied and P_2 is not satisfied?". If this query has no solution then policy P_1 implies P_2 for all possible subjects and objects. These ideas will be illustrated in the following sections.

5. The realization of semantic anchoring

This section is devoted to description of how the models are mapped to the FORMULA language. Each syntactic construct of FORMULA can be mapped to a well-defined mathematical structure presented in the previous section. This means that when we create a mapping to FORMULA, indirectly we provide a mapping to mathematical constructs as well. Recall that certain semantic anchoring has already been provided in the modeling environment. However, the mapping is not complete yet. (i) The graphically represented relations, parameters and the policies composed of them still need to be anchored. (ii) The component templates are anchored to semantic templates. The template parameters of the semantic templates must be resolved in the specialized templates. In order to introduce template parameters and their specialization, new keywords have been introduced in the modeling environment on top of FORMULA code, namely, the *bind* and *unique*. They are resolved by the mode-FORMULA mapping.

However, before we present how we assign precise and formal semantics to these language elements, we summarize the most important principles. These are the lessons that we learned from mapping several modeling languages to FORMULA, and we strongly believe that it not only made it possible for us to create an efficient semantic anchoring for several modeling languages, but also serves as an indispensable guide for others who try to map their model-based paradigms to CLP languages.

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

5.1. The semantic gap

GME metamodels use class diagrams. This claim can be stated about most metamodeling tools and environments. FORMULA has domains and terms that can have attributes and references to each other. The similarity is immediately obvious: we map metamodels to domains, and classes to terms. We only need to write a model compiler that takes the metamodel of a language, and creates a FORMULA domain for it. We need to pay attention to certain constructs, such as FORMULA supports polymorphism by its union construct ($A = B + C$ means that the term A can be either B or C) but not attribute inheritance, therefore, it must be resolved. Once we have completed this model compiler, an arbitrary modeling language can be mapped to FORMULA, we do not have to provide customized mappings.

If we wanted to use FORMULA for a formal specification only, using the generic GME-FORMULA compiler for would work. This is exactly how the first version of our policy environment was realized. However, in this case we wanted to reason about the policies in our FORMULA domain, and this became extremely inefficient. In other words, FORMULA is not only a specification language, but also an execution environment, offering a more descriptive and stronger language on top of the Z3 SMT solver. In general, undecidable problems can be described in FORMULA, but FORMULA may be unable to solve them. Therefore, we had to design the mapping very carefully to have an acceptable execution time. Starting from the simplest and generic GME-FORMULA mapping outlined above, only the third version of our mapping produced satisfactory results.

In summary, it is quite tempting to perceive the gap between a domain-specific language and FORMULA as that of syntactic nature, but in fact it is deeply semantic. This means that not only a syntactic transformation must be performed between the two tools, but a semantic translation as well. Fundamentally, there are two distinct places to perform this transformation: inside or outside FORMULA. Inside FORMULA, the solution is using transformation provided by the tool. However, we have taken the other approach, since the mapping includes processing new keywords, and only future versions of FORMULA will support more extensive string operations such as the ones needed for parsing. In this case we need to describe the mapping precisely. Below we provide the formalization of this mapping.

5.2. Semantic mapping

The semantic mapping involves three main steps. (i) We need to map the graphical models to logic programming constructs. This means specializing the templates, mapping them to logic programs, and creating the terms based on the ontology models. (ii) In order to cover the semantic gap discussed in the previous section, we introduced a few new keywords in addition to the ones provided by FORMULA. These keywords must also be resolved. Although this step seems insignificant, we believe that it illustrates a rather unique and important technique to elegantly resolve the semantic gap. Then we need to resolve the specialized parameters within the FORMULA code. (iii) Defining the search space (a partial model in FORMULA) with the entities (facts) over which the logic program must be solved.

Recall that our policy language is template-based. This means that the policies are specified by generic templates ("relations") specialized by concrete parameters. Quite similarly to C++ and other general purpose programming languages, once the templates are specialized with different parameters, they must be treated as separate, non conflicting entities. This is exactly how we need to map the policies.

As a first approach, we could map relations to FORMULA queries. However, this solution does not lead to a correct mapping. The reason is that the queries with the same name in FORMULA denote OR relation between the two. In fact, it means that the search engine can try either of them to find a solution. This implies that the different specializations interfere with one another, which clearly violates our initial rule about separate entities. This problem, however, can easily be resolved with a simple name mangling. Therefore, each specialized relation is mapped a FORMULA query with a unique name. The relations can either specify logical operation between two queries (e.g. OR or AND relationship), or specialize the relations. The mapping of the former one is the appropriate combination of the queries, while the latter is resolved by substituting the parameters in the FORMULA code.

An example semantics for our case study is depicted in Fig. 13.

The *canRequest* fact must be of type *Requestor*, which is a template parameter. Similarly, *canRequestOn* and *canRequestObj* are facts of type *Subject* and *Object*, respectively. After specialization, *Requestor* is bound to *External_Provider*, whereas *Subject* and *Object* will become *Patient* and *PharmacyRecord*, respectively in our example. These entities must exists. *Constraint* is not a fact, thus no binding is necessary. After specialization it will become *truism* in Tennessee, where there is no additional constraints, and, for California, a query that ensures the treatment relationship. This query is also specified as an instantiated template.

In order to make the specialization simple to express, we introduced the *bind* keyword. This keyword can bind a template parameter to a concrete term. Again, we need to consider possible interference. In FORMULA, term definitions are global, and a term with the same values always considered the same. In order to avoid the interference, the *bind* keyword also introduces an ID unique within a semantic anchoring. If we create a function within a semantic anchoring element, we might want to isolate in order not to interfere with other specializations. This can be achieved by the *unique* keyword. Again, we emphasize that both of the discussed keywords are not FORMULA keywords, they are resolved at generation time.

The generated semantics after specialization for the Tennessee pharmacy policy is presented in Fig. 14.

The long names are due to the name mangling discussed above. The first query (*...RecordRequest*) is for the policy, the second one (*...TN$_P$harm$_P$olicy*) is for the constraint. *Truism* (defined as *true* = *true*) is the binding *No Constraint* for the

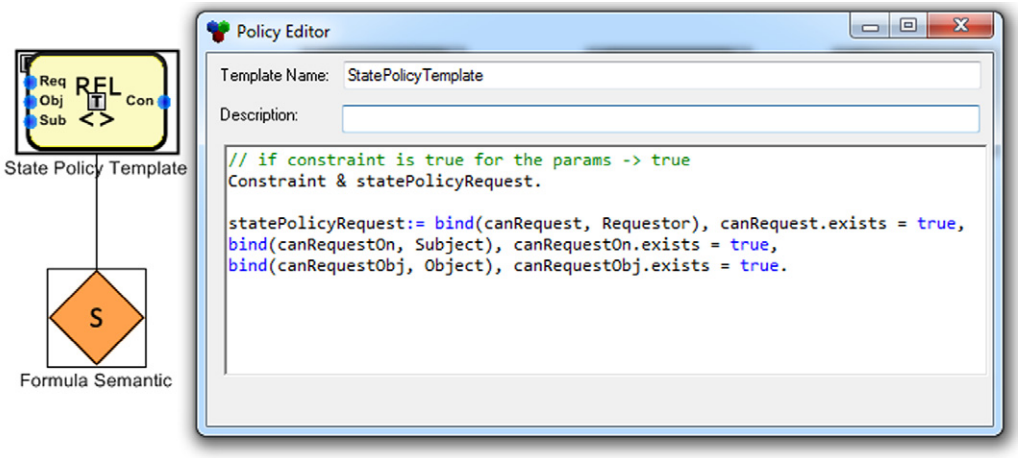

Fig. 13. The FORMULA semantics of the policy template.

```
TN_TN_Pharmacy_Record_Request := StatePolicies_TN_TN_Pharmacy_Record_Request_TN_Pharm_Policy.
            StatePolicies_TN_TN_Pharmacy_Record_Request_TN_Pharm_Policy := // if constraint is true for the params -> true
Truism & statePolicyRequest.

statePolicyRequest:= canRequest is External_Provider, canRequest.ID = "id-0067-00000068", canRequest.exists = true,
canRequestOn is Patient, canRequestOn.ID = "id-0067-00000067", canRequestOn.exists = true,
canRequestObj is PharmacyRecord, canRequestObj.ID = "id-0067-00000066", canRequestObj.exists = true.
```

Fig. 14. The FORMULA code for the Tennessee pharmacy policy generated from the policy template semantics.

Constraint parameter. The IDs have been generated automatically. The body of *statePolicyRequest* is the instantiation of the template *StatePolicy* as discussed above.

Finally, we have to generate the search space for the logic program. This is the most important step. We had multiple options at this point. Again, these options are closely related to the semantic gap. FORMULA is the most efficient, when it must resolve unknown attributes, as opposed to the case when it needs to add terms to the search space. The nature of the privacy policies are such that they require the existence of a certain document, relationship, or a qualified stakeholder. Instead of defining the conditions for the existence of certain terms, the search space contains all the possible terms that can be available. Each term has a Boolean attribute that marks the existence of the term. If it is *true*, the term is considered existing, otherwise it is not counted as existing. This appears in the constraint definition, where the *exists* property of a term can be included in a constraint. This not only results in more efficient execution, but also makes the code more reliable. In the following, we show how the pharmacy record request policy can be translated to terms and logic program.

5.3. Formalization of semantic mapping

When implementing the semantic mapping, we have used our recently developed semantic anchoring method [43]. In this, we distinguish the production workflow, and the formal specification workflow. These workflows are created based on the following process. (i) First, the specification workflow is created, i.e. the semantic mapping is specified in FORMULA. (ii) Since the tool is able to execute the specification, we first test our rules—this helps to filter the trivial mistakes, and the developer performs unit tests. At this point, there could still be both conceptual and implementation errors. (iii) Therefore, we specify formal validation requirements for the mapping and additional constraints in the target domain. (iv) We use the bounded model checking functionality of FORMULA: we try to make the tool generate objects in the semantic domain that violates the constraints in order to find conceptual and implementation errors. (v) Then, based on the formal specification, we create the production workflow. The reason for the last step is twofold: (i) we can achieve better performance with imperative languages, and (ii) FORMULA is not open source, which is a prerequisite in many of our projects. Undoubtedly, the this step might introduce additional errors. However, we can still run the executable semantic specification individually for safety critical models. In the following, we illustrate this process with excerpts from the original FORMULA transformation, which can be found at http://tinyurl.com/sharps-formula-analysis.

In our model translation scenario, the most complicated part is to transform the specialized templates.

```
1  domain LogicQueries
2  {
3      primitive Query ::= (name:String, objectID:String).
4      primitive AtomicQuery ::= (expression:String).

6      Queries ::= Query + AtomicQuery.

8      OpOrQuery ::= Queries + Op.
```

```
10      primitive Un    ::= (op: UnOp,  arg: Queries).
11      primitive Bin   ::= (op: BinOp, arg1: Queries, arg2: Queries).
12      primitive Decompose::= (arg1: Queries, arg2: OpOrQuery).


15      UnOp ::= { Not }.
16      BinOp ::= { Or, And }.
17      Op ::= Un + Bin.

19      ...
20      notResolved := q is Query, no Decompose(q,_).
21      conforms := !notResolved .
22  }
```

Listing 1. Logic queries domain.

Listing 1 displays the semantic domain, which represents the logical hierarchy of logic queries. This illustrates the power of FORMULA that it can model its own query structure. The queries can be composed of other queries or atomic queries. The actual description of the atomic queries are processed as described earlier in Section 5.2. We formalized this translation with regular expressions. Since they are axioms – they define our new keywords – we do not need to validate them with bounded-model checking. Therefore, we treat the *expression* of an *AtomicQuery* as an uninterpreted string from the logic program's perspective. The queries are connected with unary operators (NOT) or binary operators (OR, AND). The queries must be resolved by either other queries or an atomic query. This is expressed by the *notResolved* query. Note that much more constraints can be added to check the solution. Furthermore – we did not apply this technique in this project – we can automatically create a correspondence/tracing domain during the transformation to express and check even richer verification properties on the semantic domain.

As a second step, we have to create a FORMULA representation of the GME metamodel and model. We have developed a tool for this task. Since this is involves only changing the representation of the metamodel/model, it can be thought of as a simple GME exporter. Since it has the same concepts as the GME metamodel, we do not list this domain. The next step is the transformation, which is presented in Listing 2

```
2   transform T from in1::Policy_Ontology to out1::ASD
3   {
4       ...
5       ObjectRelationships ::= (or:ObjectRelationship, parent:PolicyOrConstraint).
6       ObjectRelationshipFill ::= (or:ObjectRelationship, fill:Fill_Source).
7       ObjectRelationshipSemanticAnchoring ::= (or:ObjectRelationship, semanticExpression:String).
8       ...

10      out1.Query(parent.name, parent.objectID),
11      out1.Query(name, objectID),
12      Decompose(out1.Query(name, objectID), AtomicQuery(semanticExpression)),
13      Decompose(out1.Query(parent.name, parent.objectID), out1.Query(name, objectID)) :-
14                  ObjectRelationships(instanceOr, parent),
15                  instanceOr is in1.ObjectRelationship(name, _, objectID, _),
16                  no ObjectRelationshipFill(instanceOr, constraint),
17                  constraint is in1.Constraint,
18                  ObjectRelationshipSemanticAnchoring(instanceOr, semanticExpression).

20      ...
21  }
```

Listing 2. Formalized semantic mapping.

The transformation starts with definitions: these are similar to primitives, except that they will be instantiated by the transformation engine. They can be thought of as temporary maps used during the transformation only. These are automatically filled out by the rules not listed here. The rather complex rule is better to be analyzed from the :- sign. The right hand side (RHS) of the rule is a logic expression, which must be matched. For each match, the matched variables are substituted in the left-hand-side (LHS) expression, and it a fact is generated for by logic program. In our example, RHS matches all *ObjectRelationships* (two hierarchical concepts: a policy template and its attached *FormulaSemantics*) in the input model *in*1 that do not have constraints attached. From their matched parameters, two queries are created: a general *Query* for the template, and an *AtomicQuery* for the *FormulaSemantics*. After completing and unit testing the transformation, we can use FORMULA to solve the transformation for a bounded input model to check if the transformation is able to generate models that contradict to the verification constraints—in our example, it generates queries that do not resolve to atomic queries. It can be either fixed by adding well-formedness constraints to the domain/metamodel, or by correcting the transformation.

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

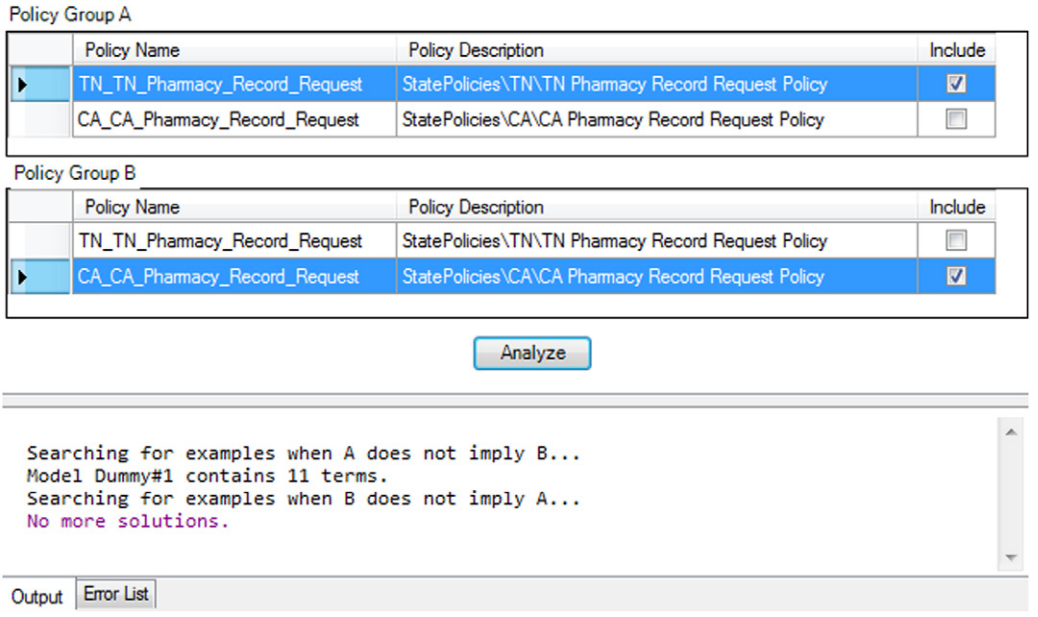

Fig. 15. The policy analysis tool and its output for the use case policies.

6. Analyzing policies

Since we validate the policies on a mathematical basis, we can reason about the policy design. If the validation of the policies are performed with the same, formally underpinned infrastructure, we can certify that the policy validation is correct by design, no additional verification is necessary.

Since we provided a formal anchoring of the policies, the verification by a logic programming tool is rather straightforward: it retrieves true or false for a specific data set passed to query certain information. Our environment can provide much more: it is able to compare two or more policies. This section describes the latter functionality provided uniquely by our tool set.

With the completed semantic anchoring of the policy templates, the formal policy models can be analyzed. To be able to analyze them, all the information that the analysis tool requires must be extracted and compiled from the models. GME provides an interpreter framework that helps in traversal and composition of the components of the models. A simple interpreter was developed that is able to perform this composition and run the analysis as described in Section 5.2.

To demonstrate how the reasoning capabilities of the PATRN framework on the formal policies using FORMULA work, we take the two policies from Section 3.2. Recall that they differ from each other only in how restrictive they are on the disclosures. Specifically the policy from California is stricter than the Tennessee one. The interpreter has a simple UI (Fig. 15) where the users can select policies into two groups that are compared to each other during the analysis. In our example, *Policy Group A* is the Tennessee policy and the California policy will be assigned to *Policy Group B*. After the analysis has been performed the results are displayed under the group assignments.

To interpret the results of the analysis, the inner mechanisms of the search algorithm of FORMULA needs to be understood. To analyze the policy sets for entailment, the two sets are injected into FORMULA with a presumption that one of the two sets implies the other. This analysis is done with the two permutations: *A implies B* and *B implies A*. FORMULA tries to find a counterexample that does not satisfy the presumptions. In order to find contradictions, the presumptions must be negated. The originally presumed implication is correct when terms are found that does not satisfy the negated presumption. From the answers of the analysis of the two negated permutation, it can deducted whether the policy set *A* implies policy set *B*, set *B* implies *A*, sets *A* and *B* are equivalent or if sets *A* and *B* are non-comparable. In the example shown in Fig. 15, the analysis found contradicting terms for negated *A* implies *B* but found no contradicting terms for negated *B* implies *A*. The answer is that *A* implies *B*, which shows that the California policy is more restrictive then the Tennessee policy.

7. Validation

The Policy Modeling environment presented in this paper is currently used in several projects, including ones that are already ongoing and ones that are in the phase of initial development. All these projects are related to the development of Health Information Exchange systems with different goals and areas of coverage. Below we present two of the most significant cooperations that leverage our policy modeling to provide its functionality.

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

Policy enforcement. The first cooperation is a health information exchange effort required as part of the formation of a clinically integrated network. The clinically integrated network collectively provides care for approximately 2 million people receiving care throughout central Tennessee. The network will consist of at five different hospital systems and numerous ambulatory care centers. Over 1500 physicians and a large number of ancillary staff will access clinical data. Almost 100 care coordinators and other professionals will be managing transitions in care and chronic illness through a $19m CMS Health Care Innovation Award. Each organization has different organizational structures, electronic health care systems, and policies. Yet of necessity this network must exchange data to measure the quality of care and to improve care across organizational boundaries. The need for consistent care coordination, the complexity of data access scenarios and policies are addressed by policy modeling, validation, and enforcement systems.

Because institutional policies can vary, each policy needs to be modeled and compared in an effort to unify these policies and assure policy execution consistent with norms set by the overarching clinically integrated network organization. As policies are added or changed, they will have to be checked to ensure they do not conflict with policies already in operation.

Using the policy modeling approach we were able to demonstrate to the stakeholders the unified policy set and grantee that the same policy will be enforced at run-time. To enable the enforcement of the same policies in different systems we leverage the flexibility provided by semantic anchoring. We have implemented the semantics of the templates in FORMULA, ILOG and Drools. The unified set contains seven high level policies containing multiple constraints for disclosure and access of electronically stored data. These policies and constraints were constructed using twelve different templates, with each template assigned with analysis semantics in FORMULA and execution semantics in both ILOG and Drools. The ontology that is used to create the policies and templates has around 250 concepts with around 560 inheritance relation connection between the elements.

Use case analysis with policies. The same technique will be applied to ensure that specialized laboratory and knowledge management services supplied to referring physicians are applied consistently. In this use case the policy modeling enables us to provide assurances that the planned new business relation with the external labs falls under the current institutional and state policies for treatment. To be able to achieve or goals we have developed a small use-case modeling language where we were able to describe the information flow required to the newly added business relation. The use case language has its semantics written in FORMULA with the models also translated into FORMULA. To fuse the policy models with use cases in FORMULA, we have developed use-case test semantics – also written in FORMULA – for a subset of the templates already developed for the HIE project. The semantic anchors for all the used templates together takes only around 60 lines of code, the generated fused analysis FORMULA model, however, has 350 lines of code from which the generated policy rules take up a roughly 100 lines. In this case the number of policies tested against the use case was fairly limited it still shows a significant reduction in the code base that needs to be maintained.

Our collaborators previously had policies in textual programs written by software engineers, especially Java and various script languages. Our environment facilitates users of various levels of expertise—none of them requires software engineers. The scenario demanding the most expertise requires writing very simple logic programs: stating the existence of ontology entities. The template mechanism allows users with no programming expertise to write policies. Following our case study, having the *state policy template* ready, no programming skills are needed to specify the Tennessee and California pharmacy policy. On this level, the only required knowledge is using GME. It has been proven over decades that domain experts of various – many times non-engineering – background can learn this without difficulties.

During the course of the two highlighted projects we were working in a very diverse team that included on one end of the spectrum hospital decision makers, mid-level managers, domain experts and undergraduate students interns and computer scientist on the other. While all the health care professional were part of the informatics core and some of them even had formal computer science training they were not programmers. The team include student interns including students studying computer science with some programming knowledge and students from outside of engineering having knowledge on philosophy and, in an instance, economics. In Table 2 we show how our team of diverse individuals were able to divide and conquer the formalization problem.

It is really hard to measure how simpler the formal description of policies have become by applying Model-Based Engineering as opposed to formal textual specification languages, since it varies with the individuals. While working on the mentioned projects, we have observed that computer scientists and programmers prefer textual representations. On the other hand, non-computer scientist domain experts feel more comfortable with graphical tools. Our experience supports the claim that MDE is a tool for domain experts who are not good at programming and learning new textual languages, but are used to the user interface of the modern computer. We believe that the real success of MDE in our projects is similar to that of the graphical user interface. The question is not if a user or is able to click or write a script quicker. It simply opens the world of computer automation – including formal problem description and analysis – to people who could not have access to it otherwise.

Our solution is not only scalable in the level of expertise. The size of policy specifications are by far not the most demanding for GME. For example, Cyber-Physical System models such as infantry vehicle models require much more model elements. GME has successfully met the demands of these applications. In case of the DSML used by the PATRN tool, the GME type-instance mechanism makes larger policy models a well-organized a conceivable by humans. The computational performance is provided by FORMULA. In general the algorithm is exponential with the number of facts that can exist. How-

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

Table 2
Tasks related to policy formalization and the expertise they require, including a brief explanation on how it was accomplished in the projects.

Task	Required expertise	Expertise used in projects
Extracting and collecting policies	Knowledge on policies and legal frameworks	The collected and extracted policies where provided to us by our cooperators that includes institutional lawyers and sociologists researching the effect of regulations.
Translating policies into formal models	Limited domain knowledge, knowledge of the representation language	After the initial development and testing of the template languages the task were transferred to the project interns and domain experts, none of them computer scientists.
Creating formal template models	In-depth domain knowledge and knowledge on creating modeling languages	This task were done by the authors for the mentioned projects.
Creating formal semantics for templates	In-depth knowledge of the target environment and in depth knowledge of the modeling language.	This task were done by the authors for the mentioned projects.
Model curation and quality assurance	Some knowledge on the representation and knowledge on the formalized policies.	The task of curation was iteratively performed by a subgroup of our project team that included modelers of the policies, which in most parts were the project interns, computer scientists with architectural knowledge and domain experts.

ever, in case of pairwise comparison, this number can be considered a constant, which makes the comparison polynomial. Thus, ordering the policies along a lattice makes the comparison more efficient for our typical consistency analysis problem.

8. Conclusion

We have discussed a model-integrated authoring environment that supports creating and analyzing privacy policies. We have addressed the following challenges. (i) The policies are described using legal and medical concepts and terminology. (ii) Both the description of the policies and the inference about them requires formal representation of policies written in human languages. (iii) Besides the component of the domain-specific environment, the tool is required to be fully automated in order for legal and medical experts to use it.

The presented model-integrated computing environment and approach delivers novelties in several areas. By separating the models and semantics it enables separation of concerns in modeling the formal policies and integrating the policies into any formal framework using semantic anchoring. With this divide and conquer approach the creation of formal policy models became easier and less error prone because of the correct by construction principle that templates enforce. The template formalisms also enable easier use of NLP in the future as the templates can provide both structure and context for the processing. With the use of NLP the creation of the policy models will be further simplified. On the other end of the divide and conquer the integration of the policy model into a formal framework has also been simplified using semantic anchoring. The implementers of the semantics for the templates do not have to have deep domain knowledge to be able create correct semantics. The knowledge they need to have is on the template language and knowledge of the system or framework the policies are compiled into. This separation also enables the policy experts and semantic developers to work concurrently and deliver end-to-end formal models quicker.

In order to further improve the applicability and usability of the PATRN framework we plan to incorporate a few changes and additions. To further improve the reusability of the policy models a policy model repository is in development. This repository will enable the sharing, reuse and crowd sourcing the policy templates as well as the policy models. We are also in the process to extend the user base of the PATRN framework by factoring some of the functionalities out of the GME environment into standalone products. This refactoring will enable more custom authoring and editing tools tailored to the needs of the policy authors.

This research and development effort is the part of the SHARPS (Strategic Health IT Advanced Research Projects) program with several participants, including hospitals and legal advisors. The release of the environment has already been completed, and currently tested by the SHARPS partners. The long term expectation is that this information system-independent policy authoring environment will be used by policy officials to compare policies between institutions, and legal experts for the law harmonization between the levels of enforcement (federal, state, institution). Also, efforts have been made to integrate the policy verification process to health information systems to achieve correctness by design.

Acknowledgments

We would like to thank the following people for their help with the research and engineering work. We would like to thank Dr. Mark E. Frisse for his insight on HIEs and the state of the Health Care in general. We would like to thank Janos L. Mathe for his help on the OWL-based Ontology language.

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

The work presented in this paper was funded through National Science Foundation (NSF) TRUST (The Team for Research in Ubiquitous Secure Technology) Science and Technology Center Grant Number CCF-0424422 and Office of National Coordinator for Health Information Technology (ONC) Grant Number HHS 90TR0003/1. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the HHS or NSF.

References

[1] H. Nissenbaum, Privacy as contextual integrity, Washington Law Review 79 (2004) 119–158.
[2] A. Barth, A. Datta, J.C. Mitchell, H. Nissenbaum, Privacy and contextual integrity: framework and applications, in: Proceedings of the 2006 IEEE Symposium on Security and Privacy, IEEE Computer Society, Washington, DC, USA, 2006, pp. 184–198. http://dx.doi.org/10.1109/SP.2006.32.
[3] P.E. Lam, J.C. Mitchell, S. Sundaram, A formalization of HIPAA for a medical messaging system, in: Proceedings of the 6th International Conference on Trust, Privacy and Security in Digital Business, TrustBus'09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 73–85.
[4] A. Barth, J.C. Mitchell, J. Rosenstein, Conflict and combination in privacy policy languages, in: Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES'04, ACM, New York, NY, USA, 2004, pp. 45–46. http://dx.doi.org/10.1145/1029179.1029195.
[5] M.C. Tschantz, D.K. Kaynar, A. Datta, Formal verification of differential privacy for interactive systems, CoRR abs/1101.2819.
[6] D. Garg, L. Jia, A. Datta, A logical method for policy enforcement over evolving audit logs, CoRR abs/1102.2521.
[7] A. Barth, J. Mitchell, A. Datta, S. Sundaram, Privacy and utility in business processes, in: Computer Security Foundations Symposium, CSF'07. 20th IEEE, 2007, 2007, pp. 279–294. http://dx.doi.org/10.1109/CSF.2007.26.
[8] A. Datta, J. Franklin, D. Garg, D. Kaynar, A logic of secure systems and its application to trusted computing, in: 30th IEEE Symposium on Security and Privacy, 2009, 2009, pp. 221–236. http://dx.doi.org/10.1109/SP.2009.16.
[9] A. Datta, J. Blocki, N. Christin, H. DeYoung, D. Garg, L. Jia, D.K. Kaynar, A. Sinha, Understanding and protecting privacy: formal semantics and principled audit mechanisms, in: S. Jajodia, C. Mazumdar (Eds.), ICISS, in: Lecture Notes in Computer Science, vol. 7093, Springer, 2011, pp. 1–27.
[10] P.E. Lam, J.C. Mitchell, A. Scedrov, S. Sundaram, F. Wang, Declarative privacy policy: finite models and attribute-based encryption, in: Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium, IHI'12, ACM, New York, NY, USA, 2012, pp. 323–332. http://dx.doi.org/10.1145/2110363.2110401.
[11] D. Basin, F. Klaedtke, S. Müller, Policy monitoring in first-order temporal logic, in: T. Touili, B. Cook, P. Jackson (Eds.), Computer Aided Verification, in: Lecture Notes in Computer Science, vol. 6174, Springer, Berlin, Heidelberg, 2010, pp. 1–18.
[12] M.Y. Becker, C. Fournet, A.D. Gordon, SecPAL: design and semantics of a decentralized authorization language, Journal of Computer Security 18 (4) (2010) 619–665.
[13] R. Craven, J. Lobo, E. Lupu, J. Ma, A. Russo, M. Sloman, A. Bandara, A formal framework for policy analysis, Imperial College London, Tech. Rep.
[14] G. Hemingway, H. Su, K. Chen, T. Koo, A semantic anchoring infrastructure for the design of embedded systems, in: Computer Software and Applications Conference, COMPSAC 2007, 31st Annual International, vol. 1, 2007, pp. 287–294. http://dx.doi.org/10.1109/COMPSAC.2007.39.
[15] Y. Lu, A. Cicchetti, S. Bygde, J. Kraft, C. Norstrom, Transformational specification of complex legacy real-time systems via semantic anchoring, in: Computer Software and Applications Conference, COMPSAC'09. 33rd Annual IEEE International, Vol. 2, 2009, 2009, pp. 510–515. http://dx.doi.org/10.1109/COMPSAC.2009.184.
[16] D. Balasubramanian, E. Jackson, Lost in translation: forgetful semantic anchoring, in: Automated Software Engineering, ASE'09. 24th International Conference on IEEE/ACM, 2009, 2009, pp. 645–649. http://dx.doi.org/10.1109/ASE.2009.83.
[17] D. Harel, B. Rumpe, Meaningful modeling: what's the semantics of "semantics"? Computer 37 (2004) 64–72. http://dx.doi.org/10.1109/MC.2004.172.
[18] Q. Liu, L. Dou, Z. Yang, A unified operational semantics for UML in situation calculus, in: G. Shen, X. Huang (Eds.), Advanced Research on Computer Science and Information Engineering, in: Communications in Computer and Information Science, vol. 153, Springer, Berlin, Heidelberg, 2011, pp. 484–490.
[19] M. Wermelinger, T. Margaria (Eds.), Fundamental Approaches to Software Engineering, 7th International Conference, FASE 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004 Barcelona, Spain, March 29–April 2, 2004, Proceedings, in: Lecture Notes in Computer Science, vol. 2984, Springer, 2004.
[20] P. Cousot, Formal verification by abstract interpretation, in: A. Goodloe, S. Person (Eds.), NASA Formal Methods, in: Lecture Notes in Computer Science, vol. 7226, Springer, 2012, pp. 3–7.
[21] S. Weirich, D. Vytiniotis, S.L.P. Jones, S. Zdancewic, Generative type abstraction and type-level computation, in: T. Ball, M. Sagiv (Eds.), POPL, ACM, 2011, pp. 227–240.
[22] C.A. Gunter, Semantics of Programming Languages: Structures and Techniques, MIT Press, Cambridge, MA, USA, 1992.
[23] K.R.M. Leino, Dafny: an automatic program verifier for functional correctness, in: E.M. Clarke, A. Voronkov (Eds.), LPAR (Dakar), in: Lecture Notes in Computer Science, vol. 6355, Springer, 2010, pp. 348–370.
[24] K. Chen, J. Sztipanovits, S. Neema, Toward a semantic anchoring infrastructure for domain-specific modeling languages, in: Proceedings of the 5th ACM international conference on Embedded software, EMSOFT'05, ACM, New York, NY, USA, 2005, pp. 35–43.
[25] Y. Gurevich, Evolving algebras: an attempt to discover semantics, 1993.
[26] A. Gargantini, E. Riccobene, P. Scandurra, A semantic framework for metamodel-based languages, Automated Software Engineering 16 (2009) 415–454. http://dx.doi.org/10.1007/s10515-009-0053-0.
[27] K. Chen, J. Sztipanovits, S. Neema, Compositional specification of behavioral semantics, in: Proceedings of the Conference on Design, Automation and Test in Europe, DATE'07, EDA Consortium, San Jose, CA, USA, 2007, pp. 906–911.
[28] A. Pan, B. Bryant, Denotational semantics-directed compilation using prolog, in: Applied Computing, 1990, Proceedings of the 1990 Symposium on, 1990, pp. 122–127. http://dx.doi.org/10.1109/SOAC.1990.82152.
[29] D. McGuinness, F. Van Harmelen, et al. OWL web ontology language overview, W3C recommendation 10.2004-03 (2004).
[30] E.K. Jackson, J. Sztipanovits, Formalizing the structural semantics of domain-specific modeling languages, Software and Systems Modeling 8 (2009) 451–478. http://dx.doi.org/10.1007/s10270-008-0105-0.
[31] E.K. Jackson, D. Seifert, M. Dahlweid, T. Santen, N. Bjørner, W. Schulte, Specifying and composing non-functional requirements in model-based development, in: A. Bergel, J. Fabry (Eds.), Software Composition, in: Lecture Notes in Computer Science, vol. 5634, Springer, Berlin, Heidelberg, 2009, pp. 72–89.
[32] E.K. Jackson, T. Levendovszky, D. Balasubramanian, Reasoning about metamodeling with formal specifications and automatic proofs, in: J. Whittle, T. Clark, T. Kühne (Eds.), MoDELS, in: Lecture Notes in Computer Science, vol. 6981, Springer, 2011, pp. 653–667.
[33] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomasson, G. Nordstrom, J. Sprinkle, P. Volgyesi, The generic modeling environment, in: Workshop on Intelligent Signal Processing, Budapest, Hungary, 2001.
[34] J. Pritts, S. Lewis, R. Jacobson, K. Lucia, K. Kayne, Report on state law requirements for patient permission to disclose health information, RTI International report.
[35] O. Lassila, R. Swick, Resource description framework (RDF) model and syntax, World Wide Web Consortium, http://www.w3.org/TR/WD-rdf-syntax.
[36] A. Ledeczi, M. Maroti, A. Bakay, G. Nordstrom, J. Garrett, C. Thomasson, J. Sprinkle, P. Volgyesi, GME 2000 users manual (v2.0), December 2001.
[37] G.K. Savova, J.J. Masanz, P.V. Ogren, J. Zheng, S. Sohn, K.C. Kipper-Schuler, C.G. Chute, Mayo clinical text analysis and knowledge extraction system (ctakes): architecture, component evaluation and applications, Journal of the American Medical Informatics Association 17 (5) (2010) 507–513. http://dx.doi.org/10.1136/jamia.2009.001560.

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004

[38] O. Bodenreider, The unified medical language system (UMLS): integrating biomedical terminology, Nucleic Acids Research 32 (suppl 1) (2004) D267–D270. http://dx.doi.org/10.1093/nar/gkh061.
[39] J. Jaffar, M.J. Maher, K. Marriott, P.J. Stuckey, The semantics of constraint logic programs, Journal of Logic Programming 37 (1–3) (1998) 1–46.
[40] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Computing Surveys 33 (3) (2001) 374–425.
[41] L.M. de Moura, N. Bjørner, Z3: an efficient SMT solver, in: C.R. Ramakrishnan, J. Rehof (Eds.), TACAS, in: Lecture Notes in Computer Science, vol. 4963, Springer, 2008, pp. 337–340.
[42] Y. Gurevich, I. Neeman, DKAL: distributed-knowledge authorization language, in: CSF, IEEE Computer Society, 2008, pp. 149–162.
[43] G. Simko, T. Levendovszky, S. Neema, E. Jackson, T. Bapty, J. Porter, J. Sztipanovits, Foundation for model integration: semantic backplane, in: Proceedings of the ASME 2012 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE, 2012, pp. 12–15.

Please cite this article in press as: A. Nadas, et al., A model-integrated authoring environment for privacy policies, Science of Computer Programming (2013), http://dx.doi.org/10.1016/j.scico.2013.05.004