Using OWL and SWRL to Represent and Reason with Situation-based Access Control Policies

Abstract

Access control is a central problem in confidentiality management, in particular in the healthcare domain, where many stakeholders require access to patients’ health records. Situation-Based Access Control (SitBAC) is a conceptual model that allows for modeling healthcare scenarios of data-access requests; thus it can be used to formulate data-access policies, where health organizations can specify their regulations involving access to patients’ data according to the context of the request. The model's central concept is the Situation, a formal representation of a patient's data-access scenario.

In this paper, we present the SitBAC Knowledge Framework, a formal healthcare-oriented, context-based access-control framework that makes it possible to represent and implement SitBAC as a knowledge model along with an associated inference method, using OWL and SWRL. Within the SitBAC knowledge framework, scenarios of data access are represented as formal Web Ontology language (OWL)-based Situation classes, formulating data-access rule classes. A set of data-access rule classes makes up the organization's data-access policy. An incoming data-access request, represented as an individual of an OWL-based Situation class, is evaluated by the inference method against the data-access policy to produce an ‘approved/denied’ response. The method uses a Description Logics (DL)-reasoner and a Semantic Web Rule Language (SWRL) engine during the inference process. The DL reasoner is used for knowledge classification and for real-time realization of the incoming data-access request as a member of an existing Situation class to infer the appropriate response. The SWRL engine is used to infer new knowledge regarding the incoming data-access requests, which are required for the realization process.

We evaluated the ability of the SitBAC knowledge framework to provide correct responses by representing and reasoning with real-life healthcare scenarios.

Keywords: Access control, Conceptual model, Knowledge model, Knowledge framework, OWL, SWRL

1. Introduction

In recent years, advanced technologies that enable organizations to collect, store and manage massive amounts of data in digital form have been developed. Sometimes, the data maintained within an organization's databases is considered to be confidential. This is the case for patient health records, which may hold sensitive information. In such cases, the organization is responsible for establishing a data-access policy in order to maintain confidentiality.

In this paper, we present a formal access-control framework that enables organizations to define and carry out confidentiality-preserving data-access policies. In particular, we focus on healthcare organizations and health data stored in electronic health records (EHRs).

The US Health Insurance Portability and Accountability Act (HIPAA) [1], which established the first comprehensive federal rule protecting the privacy of health information, defines confidentiality as "the property that data or information is not made available or disclosed to unauthorized persons or processes". Following this definition, preserving confidentiality involves restricting access to data through, e.g., authorization and access-control models. A leading access-control model is the Role-Based Access Control (RBAC) model [2][3]. RBAC places the role of the data requestor at the center, giving holders of certain roles in an organization authorization to access particular data resources. One of the significant
advantages of RBAC is its simplicity. However, this simplicity limits the expressive power of the model, as shown by an extensive qualitative study [4] of real-world scenarios involving requests for patient data. Based on an analysis of 127 elicited scenarios, the authors found that medical personnel in certain roles were prevented from accessing required data in some scenarios. However, providing a role with unrestricted access cannot be considered a solution, as access may not always be appropriate and may violate the confidentiality of the patient’s data.

Recognizing the limitations of the RBAC model, in particular its inability to represent contextual factors (e.g., the location of the data requestor, the status of the patient, and the time of access), a different conceptual approach for access-control was proposed, called Situation-Based Access Control model, or SitBAC for short [4], is a context-based, healthcare-oriented approach. It is not an extension of RBAC, as it is based on the premise that the role is not the only dimension which must be considered in data-access scenarios. Rather, it offers a new model whose expressive power is derived from a specification language rich in domain concepts (including dimensions of context) and relationships. Specifically, the new conceptual model relies on various abstractions that aggregate the domain concepts identified during the qualitative analysis in [4]. These abstractions can be used to model data-access scenarios and thereby enable the modeling of data-access rules. The central abstraction in SitBAC, Situation, is a structured representation of a data-access scenario. The other abstractions model the Entities involved in data-access scenarios (namely, Patient, Data-Requestor, Task, Legal-Authorization, EHR, and Response); their Refineables (properties); and the Relations among them. The legal-authorization entity is an abstraction that serves Situations where several interoperating health organizations are required to share data.

This paper presents a formal access-control framework based on the conceptual SitBAC model [4]. With SitBAC, a set of contextual Situations, consisting of abstractions for entities, refineables and relations, can be defined to formulate a shared knowledge model, or ontology. An ontology specifies commonly agreed, content-specific definitions for the sharing and reuse of knowledge [5][6][7]. Ontologies define a common vocabulary of the entities or concepts that are presumed to exist in some area of interest, their attributes, and the relationships that hold among them. As Strang [8] put it, ontologies are the most expressive models for designing context-aware systems. As SitBAC is an entity-based conceptual model, representing its knowledge model as an ontology was natural.

Following the above, we called our framework the SitBAC Knowledge Framework, we chose the Web Ontology Language (OWL) [9] as our ontology representation language, and we used the Protegé [10] knowledge-modeling tool to specify the SitBAC ontology. We chose OWL for two main reasons: (1) OWL was designed for sharing information over the web, meaning that a group of healthcare organizations can use it to define and share a common data-access policy by creating a set of data-access rule classes (represented via Situation classes); and (2) OWL has a built-in classification inference method from which the response to a data-access request can be inferred. Being a Description Logics (DL) [11] language, OWL can use a DL reasoner. The DL reasoner classifies the data-access rule classes (Situation classes) and realizes an incoming data-access request (represented via an individual of a Situation class) as a member of a data-access rule class, from which the appropriate ‘approved/denied’ response is inferred. Because the data included in the individual is insufficient to accomplish the realization process, we also use a Semantic Web Rule Language (SWRL) [12] engine to infer new knowledge regarding the individual by chains of properties (e.g., the patient’s location is equal to the data requestor’s department).

The SitBAC ontology, including the formal representations for the SitBAC abstractions and the Situation classes, was designed to be minimal, complete, and non-conflicting, taking advantage of ontology

---

1 Individual is the term used in OWL to describe a member of an ontology class.
exception patterns [13] and using a DL reasoner to discover potential duplications of data-access rule classes. However, the above design does not provide full support for closed-world reasoning, which is necessary when an explicit response (approved or denied) is required. Closed-world reasoning is not naturally supported by OWL; rather, OWL uses open-world reasoning, in which it cannot assume something does not exist until this is explicitly stated. To this end, the incoming data-access requests have to be "closed" as part of the matching process.

There exist other context-aware access-control frameworks (reviewed in Section 2.2) that aim to support the same goal of confidentiality preservation through the use of context attributes and associated methods that operate on them. However, the SitBAC knowledge framework is distinct as it (1) is based on a conceptual knowledge model derived from extensive qualitative research which elicited 127 data-access scenarios from the healthcare domain; (2) captures data-access scenarios specific to the healthcare domain and represents the associated context via ontological formalism; and (3) enables the use of a DL reasoner, which is a powerful tool for real-time evaluation of incoming data-access requests.

The rest of the paper is organized as follows. Section 2 presents background and related works. In Section 3 we describe our design principles and methods. The evaluation and results of our research appear in Section 4. We discuss the findings in Section 5 and conclude in Section 6.

2. Background and Related Works

In this Section we describe the theoretical background of our work and related works. In Section 2.1 we describe the ontology languages and reasoning tools we used in creating the knowledge framework, and briefly discuss closed-world reasoning in OWL. In Section 2.2 we discuss and compare various related works. Finally, in Section 2.3 we present a detailed conceptual description of the SitBAC model.

2.1. Ontology Languages and Reasoning Tools Used in This Research

In this Section, we describe the ontology languages and reasoning tools that we used in constructing the SitBAC knowledge framework. (1) The Web Ontology Language (OWL) is used to specify the knowledge base (i.e., the ontology), including the Situation classes. (2) A Description Logics (DL) reasoner is used to classify Situation classes and to realize individuals of Situation classes. Finally, (3) the Semantic Web Rule Language (SWRL) is used for writing property-chaining rules to reason about OWL individuals (representing the incoming data-access requests).

Web Ontology Language (OWL):

The term 'ontology' means a body of formally represented knowledge. An ontology language is a formal language used to construct a knowledge model or ontology. One of the most recent developments in standard ontology languages, endorsed by the W3C, is the Web Ontology Language (OWL) [9]. Unlike frame-based ontology languages, OWL employs a rich set of operators, e.g., intersection, union and negation, and is based on a logical model which allows the use of a reasoner. A reasoner can check whether all the statements and definitions in the ontology are mutually consistent, and can also recognize which concepts fit under which definitions. OWL knowledge models consist of Classes, Properties, Individuals (members of classes), and Restrictions. Restrictions specify facts that must be satisfied by an individual for it to be a member of the class.

We used Protégé-OWL (or Protégé for short) [10], a free, open-source ontology editor and knowledge base framework, to define the knowledge model for our work. OWL provides three increasingly expressive sublanguages. In our research we used the OWL-DL, which is based on Description Logics (DL) [11].
DL is a family of logic-based knowledge representation formalisms. DL is a decidable\(^2\) fragment of First Order Logic (FOL) and is therefore amenable to automated reasoning. DL Basics includes:

- **Concepts** – unary predicates/formulae with one free variable (e.g., Person, Doctor, HappyParent);
- **Roles** – binary predicates/formulae with two free variables (e.g., hasChild, loves);
- **Individuals** – constants (e.g., John, Mary, Italy); and
- **Operators** – for forming restricted concepts and roles.

The smallest propositionally closed DL consists of:

- Concepts constructed using Booleans: \(\cap\), \(\cup\), \(\neg\) (i.e., intersection, union, and compliment) and restricted quantifiers: \(\forall\), \(\exists\) (i.e., All-Values-From, Some-Values-From)
- Only atomic roles

As an example, the following DL expression specifies persons whose children are either doctors or have a child who is a doctor.

\[
\text{Person} \cap \forall \text{hasChild}. (\text{Doctor} \cup \exists \text{hasChild}. \text{Doctor})
\]

**Description Logics (DL) Reasoner:**

One of the key features of OWL-DL knowledge models is that they can be processed by a DL reasoner. A DL reasoner provides a set of Description-Logics inference services, such as:

1. **Consistency checking:** ensuring that a knowledge model does not contain any contradictory facts.
2. **Classification:** computing the classes' subclass relations to create a complete class hierarchy.
3. **Realization:** finding the most specific classes that an individual is a member of.

In our work we used the Pellet reasoner [14], which is integrated into Protégé.

**Closed-World Reasoning:**

Reasoning in OWL-DL is considered as open-world reasoning, as OWL-DL is based on an open-world assumption, meaning that we cannot assume something does not exist until it is explicitly stated. For example, assume we know that files f1, f2, and f3 exist in directory "Files". From this information we cannot infer that these three files are the only files in directory "Files", unless this information will be explicitly stated. Hence, the open-world assumption means that the current knowledge is incomplete. However, the goal of the SitBAC inference method is to provide a 'denied/approved' response for each given data-access request, and so complete knowledge is required. Accordingly, the reasoning process is considered closed-world reasoning. In the SitBAC knowledge framework, creating complete knowledge is achieved by various methods, including "closing" individuals (see Section 3.6 for more details about closing individuals in OWL).

There has been theoretical work on extending description logics to support closed-world reasoning in the form of non-monotonic logic [15][16] or by combining description logics with a logic programming approach with negation as failure [17][18]. Since implementations of closed-world inference in OWL reasoners are in a preliminary state [19], we developed practical steps to "close" the ontology. Our solution involves a combination of ontology patterns that enable closed-world reasoning for policy-based classification using existing monotonic OWL reasoners.

\[^2\] Logics are decidable if computations/algorithms based on the logic will terminate in finite time.
Semantic Web Rule Language (SWRL):

SWRL is intended to be the rule language of the Semantic Web [12]. SWRL, which is based on OWL, allows users to write rules to reason about OWL individuals and to infer new knowledge about those individuals. A SWRL rule contains an antecedent part and a consequent part. Both the antecedent and consequent consist of positive conjunctions of atoms. Informally, a SWRL rule may be read as: if all the atoms in the antecedent are true, then the consequent must also be true.

The following is a SWRL rule expressing the inference that a person with a male sibling has a brother.

\[
\text{Person}(?p) \land \text{hasSibling}(?p, ?s) \land \text{Man}(?s) \rightarrow \text{hasBrother}(?p, ?s)
\]

SWRL rules can also use arithmetic operators and can compute the desired behavior based on the context of the individual, which could depend on a dynamic environment with multiple components. For example, OWL and SWRL rules could be used to reason based on the distance between the car and the cross-road, that with the current speed, the car cannot pass the crossroad in time and thus an alert individual should be produced [20].

2.2. Related Works

In this Section, we discuss access-control models in general and context-based access-control models in particular, with a focus on DL-based access-control frameworks. Table 1 summarizes the differences among the models and frameworks reviewed in this Section.

2.2.1 Access-Control Models

Confidentiality can be protected, among other ways, via access-control models. One of the leading access-control models is the Role-Based Access Control (RBAC) model, proposed by Sandhu et al. in 1996 [2][3]. This model is based on the “need-to-know” principle, which states that confidentiality is preserved as long as a data disclosure process occurs when necessary and minimum details are revealed. As the model’s name implies, RBAC regulates users’ access to data based on their organizational roles. Each role is associated with a number of privileges (e.g., read, write, append, delete, create, etc.).

Since Sandhu proclaimed his model, the issue of access control has been discussed in a wide range of fields that share a common requirement: gathering sensitive information that need to be protected. An example of a relatively new area that references the issue of access control is Business Intelligence (BI). BI applications usually involve access to information in a company’s data warehouse with different levels of sensitivity. Although there exist mechanisms to define access control policies in database systems, these approaches are limited to defining policies on base tables that do not use materialized views, which are the engine behind BI applications, as they provide faster response time hence better performance. Bhatti et al. [21] provide a middleware-enabled policy-based framework that addresses the issues of authorization and access control in materialized view management.

2.2.2 Context-Based Access-Control Models

The role-based model grants (or denies) a role-player access to data regardless of the context of the request. Yet the context of the request should often affect access control. For example, with respect to the healthcare domain, the location of the entity requesting the data (e.g., at home or in the intensive care unit), the relationship between the data requestor and the patient (e.g., gynecologist or family doctor), and the time shift of the request could all affect the decision to grant or deny a data-access request. According to Baldauf and Dustdar [22], these contexts are classical characteristics of context-aware systems.

Motta and Furuie proposed the Contextual RBAC (CRBAC) model [23], which extends RBAC and enables modelers to construct data-access rules. The data-access rules specify the conditions that must be
met before access to a patient's data can be granted. A data-access rule is defined by the 5-tuple \(<Role, Privilege-Type, Operation, Object, Authorization-Type>\). For example, the proper 5-tuple \(<Secretary, +, View, Identification-Section, strong>\) means that a secretary is allowed to view data in the EHR’s identification section. The privilege-type element can be expanded from a Boolean primitive into a logical expression composed of \textit{context variables} connected via logical operators. The logical expression \((\text{aPatCod})\{\text{aPatCod in patCtx.in_patients}\}\) returns ‘true’ if the patient’s code is included in the list of hospitalized patients. CRBAC increased the expressive power of the basic RBAC model and refined its representation ability. However, modelers experienced difficulties when they tried to use CRBAC logical expressions for modeling complex data-access rules [24].

Another proposed contextual extension of RBAC is the \textit{Attribute-Based Access Control} (ABAC) model [25]. The ABAC model relates to the attribute of the subject and to the services of the object, and is suitable for an identity-less system, where subjects are identified by their characteristics. The authors present a framework that uses \textit{Sets} as data structures with their algebraic operations (i.e., \(\cup, \cap, \neg\)) to specify attribute-based policies. Because policies are written as Constraint Logic Programs (CLP) with recursion, they terminate as logic programs with \(\text{NP complete}\) run time. Conflict resolution and default policies can be specified, thereby ensuring the consistency and completeness of policies.

Carminati et al. [26] proposed a discretionary access control model that enables to control the sharing of information in web-based social networks. Similar to the approach of Motta and Furuie, the authors adopt a rule-based approach for specifying access policies, where the context of authorized users is determined by the type, depth, and trust level of the relationships existing between nodes in the network. On top of the proposed model, the authors provided a related enforcement mechanism, which complies with a semi-decentralized strategy, where each participant is in charge of specifying and enforcing his own access control policies.

Unlike these contextual access-control models, SitBAC is not an extension of the role-based model, but instead, describes the context as a relation between different entities and their attributes. The context is considered in terms of a \textit{Situation}, a request to access healthcare data.

\section*{2.2.3 Representation Formats for Context-based Access Control Models}

In this Section, we review works that make use of knowledge representation methods to represent their access-control policies. We start with XML-based works and continue with DL-based works.

\textbf{XML-Based Access-Control Frameworks}

One of the well-known XML formats for access control is the \textit{EXtensible Access Control Markup Language} (XACML) [27], developed by OASIS (a non-profit global consortium that drives the development, convergence, and adoption of e-business standards). XACML is a general-purpose access-control policy language, which focuses on technical issues and addresses how access control can be enforced by providing syntax for managing authorization decisions. The XACML syntax enables, to some extent, the capturing of context attributes in the policies. XACML supports direct context for the subject and resource entities by using the \textit{<SubjectAttributeDesignator>} and \textit{<ResourceAttributeDesignator>} XACML elements. In addition, XACML provides the \textit{<AttributeSelector>} element, which may contain an XPath expression to represent context. Like the SitBAC framework, XACML makes it possible to (1) define an access-control policy via a policy language that is used to express who can do what and when; and (2) evaluate whether a particular access request should be allowed and describe responses to those queries. In addition, XACML defines an architecture that supports these services by providing two enforcement entities: Policy Enforcement Point (PEP) and Policy Decision Point (PDP). PEP governs access control by making decision requests and enforcing authorization decisions.
An additional work using XML as its representation format is presented by Bhatti et al [28]. The authors present a policy-based framework that extends RBAC, supports contexts and focuses on healthcare. Their framework uses an expressive context-aware specification language, the XML-Based Generalized Temporal Role-Based Access Control (X-GTRBAC), which is an extension of RBAC. In addition, the authors propose an architecture to support the enforcement of their policies. The authors indicate that one of their contributions relates to the healthcare-specific requirements they identified while analyzing five use cases, where Clinical Document Architecture (CDA) serves as a representative EHR standard. However, they note the number of examined use cases is certainly not exhaustive. The X-GTRBAC language enables the encoding of disclosure and privacy by creating logical rules, using declarative predicate-based syntax, that specify temporal and nontemporal contextual constraints.

**DL-Based Access-Control Frameworks**

Recent years have seen increasing use of DL [11], and OWL [9] in particular, for defining access control policies. One of the earliest works is Rei, proposed by Kagal et al. [29]. Rei is a policy language designed for pervasive computing applications. Rei includes constructs for rights, prohibitions, obligations, and dispensations. The authors conclude that a semantic language (like OWL) is required for the representation of policies. They developed a policy engine that reasons over policies, described in the Rei policy language, and uses the knowledge thus acquired to make security decisions about access rights and obligations. The policy engine is developed in Java and uses Prolog as a reasoning engine. Another recently published work of Kagal and Pato [30] presents a new angle for maintaining privacy via semantic policy tools. In [30], Kagal and Pato describe an architecture layer that includes a policy reasoner, which examines a query prior to its execution on the database. The policy reasoner reasons over a set of predefined policies, database meta-data, and past queries. The reasoner then infers whether the current query is compliant with the defined policies and provides an appropriate justification. If the query is compliant, it is forwarded to the database for execution; otherwise the policy result and justification are returned to the requester. The policies are written in the AIR ontology–based policy language [31] and are represented in Turtle, a human readable syntax for RDF.

The strength of OWL, besides its being a semantic language, stems from its standardization and its expressive power. This is what motivated Agarwal and Sprick [32] to use DL for representing their Certification Policies (CA) framework for restricting access to web services. According to the authors, their work addresses the need for machine-understandable specification of certification policies. As they use DL for policy representation, any DL reasoner can be used for maintenance of CA.

Shields et al. [33] use semantic rules for defining access-control policies for web services. The authors present a security framework with a knowledge base implemented as an OWL ontology. The ontology includes the description of the information being protected, and SWRL-based rules to define the access rights of individuals. The authors use an OWL-DL reasoning engine to evaluate the rules. The access-control policy mainly concerns the web-service endpoint, which is a central element in a web service, and can be one of the following options: (a) requester is granted full access; (b) requester is refused any access; or (c) requester is granted limited access.

An integration of the classic RBAC model into OWL was suggested by Finin et al. [34]. They propose representing the RBAC model via a collection of OWL ontologies (ROWLBAC model) that can be used to specify and implement access-control systems. The ROWLBAC model includes a basic ontology composed of classic RBAC concepts, such as subjects, objects, roles, role assignments, and actions. On top of this basic ontology, an additional domain-specific ontology is defined, including the roles, actions, subjects, and objects of the domain, along with their relations and attributes to specify which actions are permitted, prohibited, or obligatory. Based on their proposal, Finin et al. [34] suggest extending the ROWLBAC model to support the more general model of attribute-based access control [25]. However, their DL reasoning usage is limited, as they use a reasoner not to realize instances, but rather to explore
how policies might evolve given permissions from different roles to change policies. For enforcing other aspects of the RBAC model, such as static separation of duty and dynamic separation of duty constraints, they define rules in N3Logic, which is a rule language that allows rules to be expressed in a web environment. Additional works that explore the representation of RBAC with OWL [35] have encountered similar problems, which they handle by adding rules to specify separation of duty and prerequisite constraints outside the OWL ontology.

Another important work is KAoS [36]. KAoS services and tools allow for the specification, management, conflict resolution, and enforcement of policies within specific contexts. The KAoS Policy Ontologies (KPO) are expressed in DAML (http://www.daml.org) to support the emerging semantic web. The KPO defines basic ontologies for actions, actors, groups, places, entities related to actions, and policies. A policy is a statement enabling or constraining execution of some type of action by one or more actors in relation to various aspects of some situation. Changes or additions to policies may result in conflicts between policies. To this end, the KAoS developers implemented an algorithm for policy conflict detection and harmonization. However, policy enforcement is carried out via precompiled policies, unlike the SitBAC reasoning process which realizes data-access requests in real time.

Proteus [37] is a context-aware policy framework that focuses on context and aims to handle access control in pervasive environments. The authors consider context as "any information that is useful for characterizing the state or the activity of an entity in which this entity operates", similarly to the SitBAC knowledge framework. Policies define operations on resources for each context. When an entity operates in a specific context, it automatically acquires the ability to perform the set of actions permitted/obliged in that context. The Proteus framework resolves a resource query by analyzing the relationships between the context of the query and the context of the policy, not relying on an exact match. The authors use an OWL-DL ontology and its associated DL reasoner, along with a Logic-Programming (LP)-based reasoning tool. LP-based reasoning uses context aggregation rules to support reasoning using property path relationships, and context instantiation rules to provide OWL assertions for attribute values.

The DL-based access control frameworks discussed above use a DL reasoner to maintain a consistent ontology. Within the SitBAC knowledge framework, we make similar use of a DL reasoner. However, in addition, we uniquely use it to realize in real time an incoming data-access request (the individual) as a member of one of the SitClasses.

Table 1 summarizes the differences between the various works reviewed with respect to the following criteria: 1) ontology-based, 2) specification language, 3) support contexts and extensions, 4) tools for policy specifications, 5) domain of accessed data, 6) enforcement mechanisms, and 7) reasoning support. Some of the criteria were taken from a table presented by Tonti et al. [38].

<table>
<thead>
<tr>
<th>Study reference</th>
<th>Ontology-based</th>
<th>Specification language</th>
<th>Supports contexts and extensions</th>
<th>Tools for policy specification</th>
<th>Domain of accessed data</th>
<th>Enforcement mechanism</th>
<th>Reasoning support</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBAC [2]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Generic domain</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CRBAC [23]</td>
<td>-</td>
<td>5-tuple-based propriety language, based on logical rules</td>
<td>+ Built-in domain contexts (e.g., location, time)</td>
<td>+ Healthcare-oriented</td>
<td>+ Healthcare-oriented</td>
<td>Propriety engine for maintenance</td>
<td></td>
</tr>
<tr>
<td>ABAC [25]</td>
<td>-</td>
<td>CLP language</td>
<td>Domain contexts need to</td>
<td>-</td>
<td>Generic domain</td>
<td>-</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

Table 1: Summary of differences between various models and frameworks for access control
<table>
<thead>
<tr>
<th>System</th>
<th>Domain</th>
<th>Architecture Model</th>
<th>Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XACML</strong> [27]</td>
<td>XML</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>X-GTRBAC</strong> [28]</td>
<td>XML-based propriety language, using logical rules</td>
<td>+ Built-in domain contexts</td>
<td>-</td>
</tr>
<tr>
<td>Domain contexts need to be defined</td>
<td>Healthcare-oriented</td>
<td>- architecture model</td>
<td>-</td>
</tr>
<tr>
<td>Rei [29]</td>
<td>Prolog-like</td>
<td>Domain contexts need to be defined</td>
<td>Generic domain</td>
</tr>
<tr>
<td>Shields [33]</td>
<td>OWL+SWRL</td>
<td>-</td>
<td>Generic domain</td>
</tr>
<tr>
<td>RowlBA C [34]</td>
<td>OWL+N3Logic rules</td>
<td>Domain contexts need to be defined</td>
<td>Generic domain</td>
</tr>
<tr>
<td>K AoS [36]</td>
<td>DAML+OIL</td>
<td>Graphical editor</td>
<td>Generic domain</td>
</tr>
<tr>
<td>Proteus [37]</td>
<td>OWL + LP-rules</td>
<td>+</td>
<td>Generic domain</td>
</tr>
<tr>
<td><strong>SitBAC Knowledge Framework</strong></td>
<td>OWL + SWRL</td>
<td>+ Built-in domain contexts (e.g., location, time, relations, status).</td>
<td>Uses Protégé for ontology and policy management</td>
</tr>
</tbody>
</table>

Looking at the table, we can conclude that our approach is unique in being healthcare-oriented, context-supported, ontology-based, and allowing reasoning in real time, as well as in its foundation in an extensive qualitative study that elicited a large collection of health data access scenarios for stakeholders.

### 2.3. The Conceptual Situation-Based Access Control (SitBAC) Model

The conceptual SitBAC model [4] enables structured representation of detailed scenarios involving access to patients’ data. The model was created following an elaborate qualitative study [4] that elicited various access-request scenarios from different types of stakeholders: patients, senior physicians, nurses, secretaries, and the head of the Information Systems unit. The central concept underlying this model is the **Situation**. The Situation is a structured representation of a data-access scenario. It is a pattern consisting of six **Entities** along with their **Refineables** and **Relations**, as shown in Figure 1.

The **Entities** include:
1. **Data-Requestor** – the (human) entity requesting access to the patient’s data.
2. **Patient** – the (human) entity who is the subject of the requested data.
3. **EHR** – the section of the Electronic Health Record where the patient’s data is maintained.
4. **Task** – the operation on the data that the data-requestor wishes to carry out (e.g., view medications, update prescriptions).
5. **Legal-Authorization** – a legal document authorizing the requested task (e.g., a social worker needs to view a patient’s diagnosis in order to approve payment. We discuss the legal-authorization entity in the next paragraph).
6. **Response** – the data-access decision made with respect to the situation in question: approved or denied.

One of the major advantages of the conceptual SitBAC model is its ability to cope with complex scenarios. Complex scenarios are those which involve interoperability, meaning that an entity in one organization requires access to data maintained by another organization, as can occur, for example, when a consortium of health organizations shares patients’ data. The legal-authorization entity is needed for such scenarios, many of which were identified in the qualitative research [4]. The legal-authorization entity represents all of the required factors of the application.

**Refineables** and **Relations** are defined as follows:

**Refineables**: The conceptual SitBAC model explicitly defines the set of properties – i.e., refineables – belonging to each entity. For instance, the patient entity has the refineable *age*, and the data-requestor entity has the refineable *role*. The refineables are organized hierarchically. A refineable may be assigned a value. For instance, the refineable *date of birth* may be assigned the date value `1/1/1989`.

**Relations**: A SitBAC relation can exist either between two entities or between two refineables. The third possible relation (Entity-to-Refineable) is not specified as a model element because conceptual modeling formalisms (e.g., the Class Diagram of the Unified Modeling Language (UML) [39] already have built-in
constructs for expressing such relations (e.g., a property is a component of the UML class). The SitBAC relation is characterized by a relation type, which indicates the nature of the relation (e.g., family-doctor-of-patient). The two types of SitBAC relations are discussed below.

1. **Entity-to-Entity Relation** (E2E) – There are two possible relationships of this kind: a) the EHR-patient entity pair, in which case the relation type is record-of; and b) the data requestor-patient entity pair, in which case the relation type is family-doctor-of-patient, gynecologist-of-patient, etc.

2. **Refineable-to-Refineable Relation** (R2R) – relationships between two refineables. For example, the patient’s location refineable can be related via an equal-to relation type to a data-requestor’s workplace refineable. The relation type can have one of the following values: equal-to, different-from, greater-than, less-than, and within. Figure 1 illustrates different types of Refineable-to-Refineable relations (e.g., Organization-Relation, Location-Relation).

A controlled experiment [24] evaluated the SitBAC’s usability compared with that of the competing Contextual Role-based Access-Control Model (CRBAC) [23]. The experiment compared the frame-based representation of SitBAC to a frame-based representation of the competing model. In that model, contextual relationships are specified using textual logical expressions, whereas in the frame-based version of SitBAC relationships are created by selecting concepts from constrained concept hierarchies. The two models were compared in terms of (1) specification comprehension and (2) the quality of specifications created. The results showed that understanding and synthesis of data-access scenarios was significantly better when using the SitBAC model for complex data-access scenarios.

3. Design Principles and Methods

In this Section, we describe in detail the design of the SitBAC knowledge framework. We start by presenting the development process for the SitBAC ontology and inference method (Section 3.1). In the next five Sections (3.2 – 3.6), we describe the design principles by which we constructed the knowledge model, including the creation of OWL-based basic ontology-classes (Section 3.2), the formalization of data-access scenarios into OWL-based Situation classes (Section 3.3), the formalization of incoming data-access requests into individuals of Situation classes (Section 3.4), the use of SWRL rules, which are required to infer new mandatory knowledge regarding the individuals (Section 3.5), and the process of “closing” individuals, which is required for closed-world reasoning (Section 3.6). In Section 3.7, we describe the design principles by which the knowledge model is formulated to be minimal, complete, and non-conflicting. We end this Section by briefly explaining the computation of the access-request decision (Section 3.8).

3.1. Development Process for the SitBAC Ontology and Inference Method

One of our design concerns is the nature of the inference method. Our first thought was to create a similarity-based method that would examine an incoming data-access request and search the set for the existing Situation that is most similar to the incoming request. However, such an approach is problematic because of the difficulty of deciding when an incoming request can be declared similar to an existing Situation. For example, suppose that we define a Situation as follows: a senior physician in a hospital is allowed to access data of an inpatient when the senior physician is at work. Suppose the senior physician tries to access the inpatient’s data while he is at home. The question is: is the incoming request similar to the defined Situation? The answer to this question may be non-structured and may require human intervention. Since we are dealing here with sensitive health data that needs to be kept confidential, we concluded that we should allow access only if the incoming request can be inferred from an existing Situation. By inferred we mean that the incoming request has to be either a) identical to an existing Situation, or b) a specialization (is-a, or inheritance relationship) of an existing Situation. Hence, the problem of making a denied/approved decision regarding an incoming data-access request turns out to be a Classification problem.
As classification is supported by the OWL-DL language [9], and since OWL-DL is based on Description Logics [11], we can use a DL reasoner. We designed our framework as follows: Our knowledge is represented via OWL classes, such that we represent data-access rule classes via a Situation class hierarchy. On top of this, we provide an inference method that formulates an incoming data-access request as an individual of a Situation class, and outputs an ‘approved/denied’ response. During the inference process, the method realizes the individual as a member of a Situation class, and the response is inferred from this realized Situation class.

The design principles by which we developed the SitBAC knowledge framework can be summarized as follows:

1. We formalized the conceptual SitBAC model into an OWL-based SitBAC ontology, which we created using Protégé [10]. Within the SitBAC ontology, we defined basic classes to represent the SitBAC model abstractions [4]: Entities, Refineables, and Relations. These basic classes are described in detail in Section 3.2.
2. Within the SitBAC ontology, beyond the basic classes, we created additional classes to represent the structured data-access scenarios as formal Situation classes. We named these classes Situation Classes, or SitClasses for short. See Section 3.3 for further details.
3. The incoming data-access request, which is the input of the inference method, has to be formalized too. We structured it into an individual of the most generic SitClass, and named this individual Situation on-the-Fly, or SitFly for short. At its early stage the SitFly has no value for its response element. More details about SitFlies appear in Section 3.4.
4. The SitFly includes various values, but is missing R2R_Relations that can be inferred from the attributes of the entities participating in a given Situation. This knowledge is required to accomplish the process of inference. To infer that knowledge, we set SWRL rules that define how to produce new knowledge regarding the SitFly’s R2R_Relations by chains of properties. More details about SWRL rules appear in Section 3.5.
5. Finally, the SitFly has to be "closed" prior to the execution of the inference method to enable closed-world reasoning (see Section 2.1). The closing process is described in Section 3.6. We require the SitBAC Ontology to be Minimal, Complete, and Non-Conflicting. In Section 3.7.1 we explain these definitions and in 3.7.2 we describe the methodology for creating such an ontology.
6. We end this Section by describing the composition of the inference method in Section 3.8.

3.2. Formalizing the Conceptual SitBAC Abstractions as OWL-Based Concept Classes

In the process of representing the conceptual SitBAC model (see Section 2.3) into the formal OWL-based SitBAC ontology, we used the following design principles.

1. We defined the three basic SitBAC abstractions (i.e., Entity, Refineable, and Relation) as super-classes in the SitBAC ontology, since all SitBAC abstractions were defined in [4] as types of these basic abstractions. The respective defined super-classes are: Entity, Refineable, and Relation.
2. We defined the six SitBAC entities as subclasses of the super-class Entity: Data_Requestor, Patient, EHR, Task, Response, and Legal_Authorization.
3. We defined the refineables as subclasses of the Refineable super-class.
4. The refineable classes are organized hierarchically in the SitBAC Ontology, and are associated with their entities via OWL properties. OWL supports two types of properties: object property and data-type property. Consequently, some refineables are associated with their respective entities via object properties (e.g., hasRole), while others are associated via data-type properties.
(e.g., hasDateOfIssue). The following are examples of defined OWL properties and their respective domain entities:

a. For Data_Requestor class: hasRole, hasWorkplace, hasShiftType and isInShift.

b. For Patient class: hasAge, hasStatus, hasInsurance, and hasLocation.

c. For Task class: hasEHRSection and hasAction.

d. For Legal_Authorization class: hasLegalAuthorizationType, hasDateOfIssue, and hasValidDateOfAccess (i.e., the period of time during which the legal authorization is valid).

e. For EHR class: hasOwnership.

5. A refineable may be assigned an Allowed Value. In the SitBAC ontology, we handle allowed values in one of two possible ways. (a) Some allowed values are subclasses of their respective refineable. For example, Senior_Medical_Secretary is a subclass of the Medical_Secretary class. (b) Other allowed values are enumerated individuals (instances) of their corresponding refineable. For example, the various options of patient status (e.g., emergencyStatus, inPatientStatus, nursingPatientStatus and outPatientStatus) are defined as individuals of the Patient_Status class. An allowed value is defined as a subclass if it can inherit refineables or relationships from another allowed value (e.g., Senior_Medical_Secretary is-a Medical_Secretary, and so can inherit access privileges of the Medical_Secretary), otherwise we define it as an enumeration.

6. Following the SitBAC conceptual model defined in [4], the various SitBAC relations are defined in the SitBAC ontology as subclasses of the Relation super-class. The SitBAC relations are grouped into two main subclasses: a) Entity_2_Entity_Relation (E2E_Relation for short) and b) Refineable_2_Refineable_Relation (or R2R_Relation for short). The following are subclasses of E2E_Relation: i) DataRequestor_2_Patient_Relation and ii) EHR_2_Patient_Relation. We define the following as subclasses of R2R_Relation: i) EHR_Relation, ii) Location_Relation, iii) Organization_Relation, and iv) Time_Relation. More specific SitBAC relations are defined as individuals of the above classes. For example, the relation R2R1.DR.Location.IsEqualTo.Patient.Location is an individual of the Location_Relation class. It expresses the fact that the data requestor’s location is equal to the patient’s location.

3.3. Formalizing Data-Access Scenarios into Situation Classes Forming a Hierarchy

Within the SitBAC ontology, we created the Situation Class to represent the generic pattern of a data-access scenario. All the specific data-access scenarios were formalized via subclasses of the Situation class with additional unique restrictions for expressing specific scenario characteristics. We named these subclasses SitClasses. The following is a list of design principles that guided us while representing the various data-access scenarios as SitClasses. The design principles are derived from the conceptual SitBAC model, and primarily from its reasoning requirements.

1. Following the definition of the Situation abstractions in [4], in the SitBAC ontology we created the most generic Situation class. The Situation class is related to its SitBAC entities (Data_requestor, Patient, Task, EHR, Legal_Authorization, and Response) via the following object properties: hasDataRequestor, hasE2ERelation, hasEHR, hasLegalAuthorization, hasPatient, hasR2RRelation, hasTask, and hasResponse.

2. The Situation class and its associated subclasses (i.e., the SitClasses) include restrictions. For example, the most generic Situation class includes a necessary and sufficient condition
≥hasTask min 1, which expresses the requirement that each Situation has to include at least one task.

3. The various SitClasses share common property values (e.g., the data-requestor in many of the SitClasses has to be in shift while accessing the patient's data). We therefore organized the SitClasses in a hierarchical structure as shown in Figure 2 (A). The hierarchical structure enables the reuse of common property values, which in our case are represented via OWL restrictions. For instance, all SitClasses expressing the requirements that the data requestor is assigned a role and is in shift are gathered as subclasses of the class "AbstractSituation_RoleInShift". We found that useful abstractions refer to the role of the data requestor, his/her work status, the patient's hospitalization status, and relationships between the data requestor and patient. The classes that define common property values without defining what the response to the access request should be are given the prefix Abstract. Figure 2 exhibits (B) the AbstractSituation_RoleInShift class, along with its condition (C): ∃hasDataRequestor some ((hasRole some Role) and (isInShift has Yes)). The condition expresses the requirement that the Situation include a data requestor who is in shift and has a role. Many different subclasses of this Situation class can be defined, which impose further restrictions on other properties.

4. The most important design principle stemming from the reasoning requirement is that each non-abstract SitClass actually structures a data-access rule class that defines the conditions for accessing a patient's data in the respective Situation. The left side of the rule class contains necessary and sufficient conditions (e.g., the data requestor has to be in shift) as well as inherited conditions (from its super-classes). The right side of the rule class, which is its inferred side, is the necessary condition from which implications are derived. In our case the implications are always the 'approved/denied' value of the response property. An example of a set of typical conditions appears in Table 1, which refers to a SitClass named "SitClass.HCA.InPatient", which is a subclass of the abstract class shown in Figure 2. This SitClass indicates the rule class of a data requestor who is assigned a healthcare administrator (HCA) role, and who is requesting to view demographic details of patient EHRs for patients whose status is inpatient (i.e., who are hospitalized), when the data requestor is in shift and the data requestor's workplace is equal to the hospitalized-department of the patient (all of this information appears on the left side of the rule). The response, given on the right side of the rule (see part 2 of Table 1), is "Approved".

![Diagram](image-url)
3.4. **Formalizing Incoming Data-Access Requests into SitClass Individuals**

The set of SitClasses described in the previous Section serves the inference method that evaluates an incoming data-access request. The incoming request is represented via an *individual* (*"instance"* in OWL) of the most generic Situation class. We named this individual *Situation-on-the-Fly*, or *SitFly* for short.

The structuring of a SitFly goes through three steps:

1. **Constructing an individual SitFly from an incoming data-access request.** The incoming data access request is in the form of an XML document, consisting of a set of data items that are to be collected from the databases of the organization (e.g., the data-requestor workplace, the hospitalized-department of the patient, etc). At this point, the SitFly does not include R2R_Relation individuals.

2. **Creation of all possible R2R_Relation individuals of the SitFly, based on the values of attributes of the entities that are part of a situation class (e.g., the creation of an *Organization_Relation* individual, which indicates that the data-requestor’s workplace is equal to the hospitalized-department of the patient).** The set of R2R_Relation individuals, added to the SitFly, is required for the completion of knowledge regarding the SitFly. Without this knowledge, the inference method of the DL reasoner will not be able to realize the SitFly into one of the existing Situation classes. More details about adding R2R_Relation individuals to a SitFly can be found in Section 3.5.

3. **Closing of the SitFly.** This step is necessary for closed-world reasoning. More details about closing individuals can be found in Section 3.6.

<table>
<thead>
<tr>
<th>Table 1: Typical conditions that appear in SitClasses, taken from the SitClass.HCA.InPatient class</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Necessary &amp; sufficient (n&amp;s) conditions</strong> (all conditions must hold at the same time)</td>
</tr>
<tr>
<td>AbstractSituation_RoleInShift_Inpatient_DRWorkingDept_EQ_PatientHospitalizedDept</td>
</tr>
<tr>
<td>The class is equivalent to its super-class, thus it inherits the conditions of its super-class. The inherited conditions are presented in the third section of this table.</td>
</tr>
<tr>
<td>∃hasDataRequestor some (hasRole some Health_Care_Administrator)</td>
</tr>
<tr>
<td>The data requestor is assigned a HCA role.</td>
</tr>
<tr>
<td>∃hasTask some View.Demographic_Section</td>
</tr>
<tr>
<td>The requested task is to view demographic details.</td>
</tr>
<tr>
<td><strong>2. Necessary condition</strong></td>
</tr>
<tr>
<td>∃hasResponse has Approved</td>
</tr>
<tr>
<td>The response for a data-access request with such conditions should be &quot;Approved&quot;.</td>
</tr>
<tr>
<td><strong>3. Inherited conditions</strong> (conditions that the SitClass inherits from its super-classes; all these conditions must hold at the same time):</td>
</tr>
<tr>
<td>∃hasDataRequestor some ((hasRole some Role) and (isInShift has Yes))</td>
</tr>
<tr>
<td>The data requestor is in shift.</td>
</tr>
<tr>
<td>∃hasPatient some (hasStatus has inPatientStatus)</td>
</tr>
<tr>
<td>The patient is hospitalized.</td>
</tr>
<tr>
<td>∃hasR2RRelation has R2Ro.DR.WorkingDepartment.IsEqualTo.Patient.Hospitalized.Department</td>
</tr>
<tr>
<td>The data-requestor’s workplace is equal to the hospitalized-department of the patient.</td>
</tr>
<tr>
<td>∃hasTask min 1</td>
</tr>
<tr>
<td>There must be at least one task.</td>
</tr>
</tbody>
</table>
An example of a SitFly is presented in Figure 3. We can observe that the data-access request deals with documenting an encounter (A). The SitFly indicates, via a R2R_Relation individual (B), that the data-requestor’s working department is equal to the patient’s supervision clinic. (This knowledge is inferred because both Individuals contain the same value of InternalOutClinic. The InternalOutClinic value is assigned to the working department of the data-requestor's individual, and to the supervision clinic of the patient’s individual). In addition, the SitFly indicates that the patient has the status of outpatient (i.e., is not hospitalized), and the data requestor is in shift and is assigned the role of paramedic. The response (C) is intentionally missing, as a SitFly expresses a new data-access request that has to be realized to the most specific SitClass in order to infer its response.

3.5. Adding R2R_Relation Individuals to a SitFly Using SWRL Rules

The second step in structuring the SitFly involves the creation of the required R2R_Relation individuals. These R2R_Relation individuals are inferred by the SWRL engine and are attached to the SitFly. The SWRL engine infers new facts regarding a given SitFly based on values of attributes of entities that participate in the SitFly and on a set of SWRL rules that are prepared beforehand. An example of a SWRL rule appears in Figure 4. The first part of the expression is the rule antecedent (Section 2.1) and the second (right-hand) part is the rule consequent. The meaning of this rule is: if the data-requestor's workplace is identical to the department where the patient is hospitalized, the SWRL engine will infer an Organization_Relation individual and will add it to the SitFly.

![Figure 3: An example of a SitFly. The data-access request involves documenting an encounter (A), a R2R relation indicates that the data-requestor's working department is equal to the patient’s supervision clinic (B), and the response is intentionally missing, as a SitFly expresses a new data-access request (C). The closing statements appear in (D).]

3.6. Closing the SitFlies (Individuals) for Closed-World Reasoning

In order to understand the necessity of closing the individual, consider the following example: The SitBAC ontology includes one SitClass named "SingleSitClass", and a complimentary class, which is the class that an individual becomes a member of if it cannot be a member of an existing SitClass. We
want to realize an incoming SitFly. There are two possible scenarios: (a) the SitFly is realized as a member of the SingleSitClass, or (b) the SitFly cannot be realized as a member of the SingleSitClass, and so has to be realized as a member of the complimentary class.

Suppose the SitFly is identical to SingleSitClass except for one element – the SingleSitClass includes the value Medical_Secretary, while the SitFly includes Senior_Researcher. Since Senior_Researcher and Medical_Secretary do not have a superclass-subclass relation, the reasoner cannot realize the SitFly as a member of the SingleSitClass. However, the reasoner cannot infer the opposite fact, namely that the SitFly is not a member of SingleSitClass. Recall, under the open-world assumption, the reasoner may assume that the SitFly is incomplete, and that in the future it may include the Medical_Secretary value.

In order to reject this option, we must explicitly state that the SitFly includes one and only one role. We do this by "closing" the individual (i.e., the SitFly) by adding the restriction hasRole exactly 1. Once this restriction is part of the SitFly, the reasoner can definitely infer that the SitFly cannot be realized as a member of the SingleSitClass. Thus, it is a member of the complimentary class, since this class is defined as the negated class of SingleSitClass (¬Not SingleSitClass).

The task of closing an individual involves adding the "exactly" restrictions to the individual. Figure 3(D) demonstrates the additional restrictions that close the SitFly in the figure. We can observe that two restrictions were added to the SitFly: hasPatient exactly 1 and hasR2Relation exactly 2.

If the SitFly is not closed (i.e., if it does not include the required closing statements), the reasoner cannot infer the not-a-member-of fact. In such cases, the SitFly is left in the generic Situation level and is not realized as a member of a SitClass with a necessary implication about the response.
3.7. **Formulating a Minimal, Complete, and Non-Conflictin g SitBAC Knowledge Base Model**

We wish to evaluate a SitFly (i.e., an incoming data-access request) based on the SitBAC ontology. Our objective is to create an ontology such that there is always a single answer for any given SitFly. By saying “always” we mean that the SitBAC ontology is complete, and by saying “single answer” we mean that it is non-conflicting. We can imagine the SitBAC ontology as a “sea with Situation islands”, as presented in Figure 5. The islands are “positive” SitClasses, i.e., representing data-access rule classes for which access should be granted. If a SitFly is realized as a member of one (or more) of these islands then its response is inferred to be permission granted (approved). The sea represents all the complementary (“negative”) SitClasses for which, by default, the response is to deny access.

In the following we provide formal definitions for each of the requirements and propose a methodology for building a minimal, complete, and non-conflicting SitBAC ontology.

### 3.7.1 Formal Definitions

Consider two SitClasses Sci and Scj, and a Situation individual Sfk.

- **We denote** Sci ≥ Scj, if Sci can be classified under Scj.
- **We denote** Sfk ≥ Sci, if Sfk is an inferred individual of Sci (realized as Sci).

**Definition 1: A Complete SitBAC ontology**

Consider a SitBAC ontology, a SitClass Sci, and a Situation individual Sfk. The SitBAC ontology is **Complete** if ∀ Sfk ∃ Sci, so that Sci ∈ SitBAC ontology and Sfk ≥ Sci.

**Definition 2: A Minimal SitBAC ontology**

Consider a SitBAC ontology and an ordered pair of SitClasses (Sci, Scj). The SitBAC ontology is **Minimal** if ∀ (Sci, Scj) and the following conditions hold:

- Sci, Scj ∈ SitBAC ontology, and Sci is different from Scj; and
- Sci not ≥ Scj.

**Definition 3: a Conflicting SitBAC ontology**

Consider a SitBAC ontology and two SitClasses Sci, Scj, and a Situation individual Sfk. A SitBAC ontology is **conflicting** if the following conditions hold:

- Sci, Scj ∈ SitBAC ontology, and Sci is different from Scj; and
- Sci is assigned an approved response, while Scj is assigned a denied response; and
- Sfk ≥ Sci and Sfk ≥ Scj.

### 3.7.2 A Methodology for Creating a Minimal, Complete, and Non-Conflictin g SitBAC ontology

In the following Sections we propose a methodology for creating a SitBAC ontology that will hold the above characteristics.

**Completeness Requirement**

The completeness requirement ensures that each given SitFly is realized as a member of at least one SitClass, as required for closed-world reasoning (see Section 2.1). To achieve this, the SitBAC ontology has to be designed so that for every required scenario, a respective SitClass has to be created. Each of
these SitClasses is associated with an approved response ("positive" SitClass). In addition, a Complementary Situation Class has to be created, which is basically a class with one restriction that indicates the following fact: if an individual (in our case the SitFly) cannot be realized as a member of one of the positive SitClasses, then it has to be realized as a member of the complementary class ("negative" SitClass). The complementary class is associated with a denied-value response.

Note that a SitFly may be realized as a member of more than one SitClass. For example, one SitClass may represent the access-control policy for the role of head of the hospital, and a second SitClass may represent the policy for a data requestor who is located in the emergency room. In the case of a SitFly created when the data requestor is head of the hospital and is located in the emergency room, the SitFly will be realized as a member of both SitClasses. However, no conflict will occur, as both SitClasses are associated with an approved-value response. Our method makes sure that only one non-conflicting answer is returned, although the realization may be to more than one SitClass.

[40] provides a list of the guidelines that we formulated for achieving a complete SitBAC ontology.

Non-Conflicting Requirement

According to the conflict definition, a conflict could occur if it were possible for one SitFly to be realized as a member of two different specific SitClasses that have different responses. This cannot happen, because all SitClasses are associated with an approved-value response except for the complementary class, which is associated with a denied-value response. Thus, if a SitFly is realized as a member of two SitClasses, its response is approved. If the SitFly is realized as a member of a SitClass and a member of the complementary class, a conflict occurs. However, such a case cannot occur, since if a SitFly is realized as a member of the complementary class, it cannot, by definition, be realized as a member of another SitClass, and vice versa.

Minimality Requirement

The minimality requirement ensures that our SitBAC ontology will not have two SitClasses such that one is classified as a subclass of the other, as this creates replications in the knowledge base. Minimality is important in order to facilitate maintenance. To this end, we use one of the DL reasoner services. The reasoner can infer subclass-superclass relations (see Section 2.10). Once we have created the SitBAC ontology, we can invoke the reasoner and check its inferred hierarchy class tree. If we find new subclass relations, we can examine them and delete the unnecessary class/subclass. The only exception that we can accept is a superclass that is an abstract SitClass (see Section 3.3). In such a case, we do not delete the abstract SitClass, as it does not hold a response value, and exists only for the clarification of the SitBAC ontology.

3.8. Computing the Decision Result

Recall that we wish to evaluate an incoming data-access request (the SitFly) based on the Situation classes (SitClasses) that are maintained within the SitBAC ontology, which is minimal, complete, and non-conflicting. In this Section we describe the inference method for evaluating SitFlies, and for providing an 'approved/denied' response.

The inference method is a composition of several programs: (1) the Protégé API [10], (2) the Pellet reasoner [14], and (3) the SWRL engine [12]. We used Protégé's API to access the kb, we used the Pellet reasoner to realize an incoming data-access request (SitFly) as a member of one of the existing SitClasses, and we used SWRL to infer the possible R2R_Relations in a SitFly. Though role chains are supported by OWL 2.0, we used SWRL operators (e.g., GreaterThan) that are not supported by OWL 2.0.

The inference method is composed of the following four steps, as presented in Figure 6. (1) Creating a SitFly, based on the data items included in the incoming data-access request, which is an XML document.
The various data items should be collected from the databases of the organization (e.g., the data-requestor workplace, the hospitalized-department of the patient, etc). At that point E2E_Relations are created as well (e.g., the fact that the data requestor is the family doctor of the patient is represented in the SitFly as a DataRequestor_2_Patient_Relation individual. (2) **Inferring SitBAC R2R_Relation individuals** applicable to the SitFly, based on the included values. For that purpose we use the SWRL engine. The SWRL engine infers R2R_Relation individuals and adds them to the SitFly, based on SWRL rules prepared beforehand. For example, if the data-requestor's workplace is identical to the department where the patient is hospitalized, the SWRL engine will infer an Organization_Relation individual and will add it to the SitFly. (3) To support closed-world reasoning, we **close the individual** (i.e., the SitFly) by adding restrictions of type hasRole exactly 1, as described in Section 3.6. (4) Finally, we execute the Pellet reasoner for the **realization task**. The output of Pellet is the SitClass that the SitFly is realized as a member of. The final output of the entire inference method, as presented in Figure 6, is the response value, inferred from the realized SitClass.

![Figure 6: The components of the inference method](image)

### 4. Evaluation and Results

In this Section, we report how we evaluated the SitBAC knowledge framework and its ability to provide responses for data-access requests. In addition, we describe time measurements that we performed for better understanding of the relation between the number of the SitClasses in the knowledge base and the run-time of the realization task. This understanding is important for predicting the scalability of our framework for large organizations.

#### 4.1. Evaluating the SitBAC Knowledge Framework and its Ability to Support Reasoning Tasks

To evaluate our approach, we developed a proof-of-concept prototype knowledge base that included 33 SitClasses (Situation classes), where 4 were defined as abstract classes and the other 29 were leaf SitClasses associated with a response value. Some of the SitClasses represent data-access scenarios for typical (non-special) circumstances (e.g., a family doctor can access the full EHR of his patients). Others represent non-typical scenarios, i.e., scenarios that express infrequent states that can be considered as exceptional cases for which access should be granted. For example, while a secretary is usually not allowed to view medical data, he/she may be allowed to view lab test information in cases where the patient is older than 75 and hospitalized in the intensive care unit.

The SitBAC ontology includes 159 named classes with 43 object properties and 18 data properties. The ontology also includes 143 individuals, of which 33 are SitClasses. In addition, the ontology includes 283
anonymous classes, composed of 97 existential classes, 120 cardinality classes, 1 minCardinality class, and 57 hasValue classes.

The knowledge base includes a set of SitClasses which represent the organizational data-access rules; thus, the knowledge base formulates the data-access policy of the organization. We tested its ability to infer the response for a set of 24 access-request scenarios collected from a thorough qualitative study [4]. Each of these data-access requests was formulated into a SitFly and given a meaningful name that implies its content. For example: SitFly01—ExpertPhysician.DocumentEncounter.InShift.OutPatient refers to a data requestor who is on shift and is assigned the role of expert physician, and the requested task is documenting an encounter with a patient who is considered an outpatient.

We defined various SitFlies to check and evaluate various issues:

1. The hierarchical role structure, which represents role policy inheritance. For example, we have a SitClass named SitClass.Atyp.ParamedicAtEmergency, which involves a data requestor assigned the role of paramedic and a patient with emergency status. In such a case the data requestor is allowed to view the patient's medical information. The SitFly: Nurse.ViewPrescription1.NotInShift.EmergencyStatus was realized as a member of the above SitClass. The SitFly indicates that a data requestor assigned the role of nurse, who is not on shift, is requesting to view prescriptions of a patient with emergency status.

The reasoner realized this SitFly as a member of the above SitClass because Nurse is a subclass of Paramedic and Prescription_Section is a subclass of Medical_Section, and the patient in both cases had emergency status. Thus, the conditions of the above SitClass were satisfied by the SitFly. Note that additional information (e.g., the shift status) did not affect the reasoner's decision.

2. The correct realization of SitFlies as members of SitClasses with approved-value response. For example, the SitFly SitFly12—OrdinaryHCA.ViewDemographic.InShift.InPatient indicates a data-access request by a data requestor assigned the role of HCA and on shift. The required task is to view the demographic details of a patient who is an inpatient. The data-requestor's workplace is identical to the patient's hospitalized-department. This SitFly was realized as a member of the SitClass named "SitClass.HCA.InPatient" (see Table 1.)

![A screenshot of the social worker SitClass](image-url)
3. The correct realization of SitFlies as members of the complimentary class. Ten of 24 SitFlies were realized as members of the complimentary class. We confirmed that all of these SitFlies failed to satisfy the conditions of any SitClass with approved response-value; thus, they could not be realized as members of these SitClasses.

In order for the reasoner to realize them as members of the complimentary class, these SitFlies must be closed by stating that the knowledge we have about their property values is all the knowledge that exists.

4. We evaluated the legal_authorization entity by means of a special SitClass that represented a typical daily scenario of a social worker. The SitClass presented in Figure 7 represents a data-access scenario of a social worker who is on shift (a) where the task is to view the medical section of the patient’s data (f). The type of legal_authorization required in this scenario is Patient_Consent (b). The social worker's employer has to be equal to the organization, which is mentioned in the legal_authorization form (d), and the social worker can view only sections that are noted in the legal_authorization form (c). Other restrictions defined in this SitClass deal with the date of access, which should be greater than the date of issue and less than the date of expiration (e). We created a corresponding SitFly to check its realization against the SitClass.

4.2. Run-Time Tests

In this Section, we examine the relationship between the size of the knowledge base, in particular the number of SitClasses, and the execution time of the reasoner while it realizes the SitFlies. To this end, we performed a test suggested by Horrocks [41] designed to predict the scalability of our SitBAC knowledge framework. The tested knowledge base included 33 Situation classes that we created based on more than 100 data-access scenarios identified in [4]. The hierarchical structure of the SitBAC ontology, and in particular, the role and EHR section hierarchies and the SitClasses hierarchy, enables reuse, meaning that we did not have to create a separate Situation class for each identified scenario. We believe that a small to medium organization may need 30 to 100 Situation classes, while large organizations may need up to a few hundred.

All the tests were performed on a 2.2 GHz Notebook with 2 GB memory, using Protégé version 3.4 beta.

Table 2 refers to the execution time of a reasoner realizing one SitFly against a knowledge base containing a collection of SitClasses. The table shows the realization time of this single SitFly as a function of the number of leaf SitClasses (i.e., not including the abstract SitClasses) in the knowledge base. The number of leaf SitClasses increases along the table, while the number of abstract classes stays unchanged (4 abstract classes). We used the same SitFly in all of the tests. It includes one R2R relation, calculated via the SWRL rules. The last column in the table indicates the average execution time of the reasoner. The average execution time is an average of three sequential executions with the same knowledge base and SitFly.

Our hypothesis was that the run time of the reasoner will increase as a function of the knowledge base size, and in particular, the number of SitClasses. Figure 8 presents a graph of the results depicted in Table 2. As Table 2 and the graph indicate, the execution time increases as a linear function of the number of leaf SitClasses, as expected. Based on these tests we estimate that the scalability of our SitBAC knowledge framework is linear as long as the size of the knowledge base is a few dozen to a few hundred (in our view, the expected size of a typical Situation knowledge base for healthcare organizations).

Table 2: Average execution time for realizing a SitFly

<table>
<thead>
<tr>
<th>No. of Leaf SitClasses</th>
<th>No. of Abstract Classes</th>
<th>Total Sit Classes</th>
<th>Average Exec Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>
5. Discussion

In this paper, we presented the OWL-based SitBAC knowledge framework, which makes it possible to formally represent data-access scenarios and to use SWRL and a DL reasoner for realizing incoming data-access requests and providing an 'approved/denied' response for them. The SitBAC knowledge framework is a formal implementation of the conceptual SitBAC model [4], which is a robust model that was developed after long and intensive qualitative research. Our implementation specifies Situations in which access to data should be approved as ontology classes with necessary implications, mimicking decision rules. While other formalisms can be used to specify decision rules regarding access requests, for example, decision trees, formalization as a knowledge model has an advantage in that it defines the semantics of the concepts involved. This is one of the reasons why the community of bioinformatics researchers uses ontologies to define their required knowledge [13].

The use of a DL reasoner in real time is unique to our work, as other research efforts [36] use a DL reasoner for pre-calculated data-access policies. We were able to use a DL reasoner for real-time inference because we established a closed-world reasoning infrastructure.

In the rest of this Section we discuss the limitations of our approach and future work.

<table>
<thead>
<tr>
<th>2</th>
<th>9</th>
<th>4</th>
<th>13</th>
<th>1.239</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>13</td>
<td>4</td>
<td>17</td>
<td>1.411</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>4</td>
<td>21</td>
<td>1.453</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>4</td>
<td>25</td>
<td>1.744</td>
</tr>
<tr>
<td>6</td>
<td>25</td>
<td>4</td>
<td>29</td>
<td>1.844</td>
</tr>
<tr>
<td>7</td>
<td>29</td>
<td>4</td>
<td>33</td>
<td>2.151</td>
</tr>
</tbody>
</table>

Figure 8: The execution time of the reasoner as a function of the number of SitClasses

Limitations

The OWL-based SitBAC knowledge framework that we suggest provides confidentiality as it complies with the "need to know" principle for data disclosure. However, there are overhead costs associated with
integration of our framework into EHR management systems. Such integration requires the design and implementation of methods for collecting required data items from organizational databases regarding the refineables of the data requestor and the patient involved in a data-access request. The development of such methods lies outside the scope of our work thus far, but is necessary for implementation within organizations. It should be noted that while property values of the data requestor and patient are necessary for all data-access requests, the collection of data items for the legal_authorization entity, which involves even greater complexity, may be necessary only in cases where an approved response cannot be calculated.

Another limitation concerns the complexity of the DL-reasoner. The DL-reasoner Pellet and the SWRL engine have high complexity (\textit{NExpTime-complete}). However, this complexity is due to the fact that DL reasoning can handle all features of the OWL-DL language, including features that we do not use in our SitBAC knowledge framework, such as transitive or functional properties. We evaluated the theoretical complexity of reasoning only with the specific features used in our knowledge framework via a calculator for complexity of reasoning in Description Logics [42], and determined it to be \textit{PSpace-complete}. This calculation is based on the knowledge model's DL expressivity, which is provided by Protégé [10]. The DL expressivity of our knowledge model is $ALCON(D)$, where $ALC$ stands for the basic DL, $O$ stands for nominal singleton classes (e.g., \{Italy\}), $N$ stands for numbered restrictions (e.g., $\geq 2hasChild$), and $D$ stands for data types. Our SitBAC ontology does not include the following features: property hierarchy ($H$), inverse properties ($I$), qualified number restrictions ($Q$), and functional number restrictions ($F$).

The theoretical complexity of our OWL-based SitBAC knowledge framework, \textit{PSpace-complete}, is very high. However, Horrocks et al. [41] reported experiments in which they tested the actual performance of the reasoner as a function of the size of the knowledge base, and showed that the actual performance was better than what was predicted in theory. Performing a similar experiment, we measured the realization time on a knowledge base consisting of up to 29 leaf Situation classes. The tests that we performed suggest that the approach would be scalable to healthcare organizations with up to a few hundred situation classes. However, this remains to be shown in practice.

Other researchers have optimized the DL-reasoner for the features of their specific ontologies, achieving better performance than the analytical complexity [43]. We could optimize our algorithm using a similar approach, or by developing our own special-purpose reasoner, if better performance becomes a necessity when we implement our framework within a healthcare organization.

DL reasoning is a powerful tool, but it comes with the associated requirement of maintaining precise ontologies, requiring domain experts who may not be readily available in all application environments, such as healthcare. However, a related work has demonstrated that undergraduate information system students have successfully represented healthcare policies using a frame-based ontology that corresponded to the SitBAC conceptual model [24].

\section*{Future Work}

Prior to developing the OWL-based SitBAC knowledge framework, a frame-based implementation of the conceptual SitBAC model has been developed. The frame-based implementation was selected mainly because of the ease of defining a knowledge model (classes and instances). A controlled experiment [24] was conducted comparing participants' ability to comprehend and represent scenarios of data-access requests via the SitBAC frame-based model as compared to a competing logics-based CRBAC model [4]. In our work we switched to OWL-based representation because we realized that the problem that the inference method should solve is a realization problem (see Section 3.1), which can be supported naturally by the OWL-DL language [9]. However, the ease of defining classes in the OWL knowledge model has not been evaluated, and this may not be as easy to use as the frame-based version. Although OWL uses logic too, we can improve its usability by providing user-friendly guidelines for creating SitClasses (not discussed within this paper) and then evaluate the usability in a controlled experiment.
using a sample of EHR administrators. Another direction that could potentially ease the task of modeling in OWL is to use tools such as TwoUse [44], which integrate UML models with OWL specifications in a coherent framework. The use of UML’s visual models in combination with OWL may make it easier for users to comprehend and specify SitBAC policies when compared to using OWL alone.

6. Conclusion

The OWL-based SitBAC knowledge framework for access control is suitable for healthcare organizations, and is also suitable for implementation as part of the semantic web. Though we developed our framework with respect to the healthcare domain and to patient confidentiality concerns, it can be adapted to other domains that require customer confidentiality, such as banking. The SitBAC knowledge framework, including its inference methods, offers an innovative approach to modeling scenarios and formulating them into a knowledge base that offers minimality, completeness, and non-conflicting responses.

References


