Fine-grained Access Control for EPC Information Services

Eberhard Grummt[1,2] and Markus Müller[2]

[1] SAP Research CEC Dresden
eberhard.oliver.grummt@sap.com
[2] Technische Universität Dresden
markus.mueller@mailbox.tu-dresden.de

Abstract. Inter-organizational exchange of information about physical objects that is automatically gathered using RFID can increase the traceability of goods in complex supply chains. With the EPCIS specification, a standard for RFID-based events and respective information system interfaces is available. However, it does not address access control in detail, which is a prerequisite for secure information exchange. We propose a novel rule-based, context-aware policy language for describing access rights on large sets of EPCIS Events. Furthermore, we discuss approaches to enforce these policies and introduce an efficient enforcement mechanism based on query recomposition and its prototypical implementation.

1 Introduction

RFID is quickly becoming a key technology for novel supply chain management applications. It enables the automatic identification (*Auto-ID*) of physical objects equipped with small transponder tags. In intra-organizational scenarios, RFID's advantages over the established bar code regarding efficiency and data granularity have been used for several years. In inter-organizational settings, the technology's main potential is to increase the visibility of goods along the whole supply chain. While gathering and exchanging object-related information does not pose a general challenge using technology available today, several security and incompatibility issues remain unsolved.

With the industry organization *EPCglobal Inc.*[3], there is a strong initiative towards overcoming incompatibilities between companies' RFID-related IT infrastructures. EPCglobal fosters open data format and interface standards. Besides tag data specifications, the most important standard for inter-organizational data exchange is the *EPC Information Services* (EPCIS) specification [8]. Software systems implementing this specification, called *EPCIS Repositories*, feature standardized interfaces for capturing and querying EPC-related event and meta data. Since EPCIS Repositories hold mission-critical, potentially confidential information, access to these interfaces needs to be limited. The EPCIS

[3] http://www.epcglobalinc.org/, EPC stands for Electronic Product Code

standard explicitly leaves the details how access control is performed to the individual EPCIS implementations [8, pp. 57–58].

In this paper, we present an approach to specify and enforce fine-grained access rights to large quantities of EPCIS event information. Our contribution is twofold. First, we introduce a novel access control policy language called AAL leveraging the structure of EPCIS data. Second, we present an efficient policy enforcement mechanism and its implementation that is based on the concept of SQL query rewriting for relational databases. To our best knowledge, access control for EPCIS data has not been addressed by scientific literature so far. The remainder of this paper is structured as follows: We present the problem statement, assumptions and challenges in Section 2. We give an overview of related work in Section 3. In Section 4, we introduce requirements and our concepts for a policy definition language and an efficient enforcement mechanism. We evaluate our results in Section 5 and conclude in Section 6, giving directions for future research.

2 Problem Statement

We investigate how access control policies for EPCIS Repositories can be defined and enforced efficiently in order to facilitate fine-grained disclosure control for RFID-based events.

2.1 Definitions and Assumptions

Let $C = \{c_1, \ldots, c_n\}$ be a set of companies, each of which operates an EPCIS Repository or an EPCIS *Accessing Application* [27, p. 41]. We assume that every $c_k \in C$ can be reliably identified and authenticated by every other company $c_l \in C$. An EPCIS Repository operated by any company c_m stores only EPCIS Events generated by this company, i.e. information gathered from remote sources is *not* integrated into c_m's repository. A *principal* can be any participating user, administrator, role, company, or system [2, p. 9]. In our context, a *user* is a principal that belongs to a company c_a and tries to access a remote EPCIS operated by a company c_b using an Accessing Application. An *administrator* is a principal that is allowed to grant and revoke access rights to an EPCIS Repository. We refer to *Access Control* (AC) as the process of enforcing an applicable Access Policy. An *Access Policy* (policy for short) is a formal specification that states which user or role has the right to access which EPCIS Events. An *Access Control Mechanism* or *Enforcement Mechanism* (mechanism for short) is a software component ensuring that relevant policies are applied to all access operations carried out by users, prohibiting or limiting disclosure if necessary.

2.2 Introduction to EPCIS

EPCIS Repositories store information about physical objects. This information is logically represented in the form of *EPCIS Events*. Generally, an event represents a change in state that occurs at a certain point in time. The EPCIS

specification defines four event types, namely *ObjectEvent*, *AggregationEvent*, *QuantityEvent*, and *TransactionEvent*. They are used to express object observations, object aggregations, object quantity observations, and connections of objects and business transactions, respectively [8, pp. 39–53]. While the internal processing and storage details may vary from implementation to implementation, an EPCIS Repository needs to provide two interfaces: The *EPCIS Capture Interface* (CI) and the *EPCIS Query Interface* (QI). The CI is used to submit new events to be stored in the repository, while the QI is used to retrieve events of interest. Both interfaces can be implemented in the form of web services. To that end, EPCglobal specifies HTTP and SOAP bindings [8, pp. 108–126].

2.3 Use Case and Challenges

To enable certain applications such as Tracking and Tracing (determining the current and all previous locations of an object), companies need to access events stored in EPCIS Repositories operated by other companies. EPCIS Events are confidential, because they can be used to infer production capacities, inventory levels, sales figures, and business relationships, among others. This is why access control is a prerequisite for the inter-organizational EPCIS deployment.

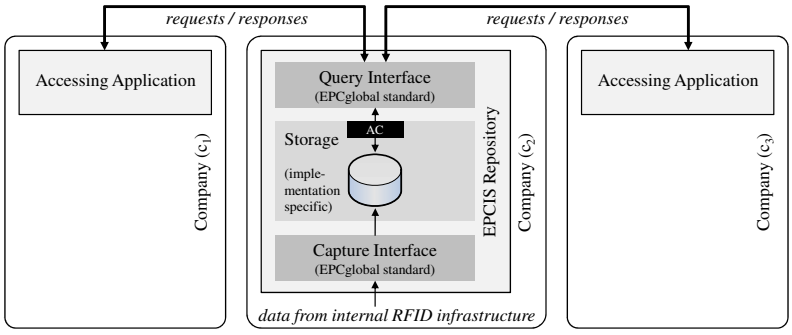

Fig. 1. EPCIS Interfaces and Interactions

Fig. 1 depicts a company c_2 that operates an EPCIS Repository. Via its Capture Interface, RFID-based events gathered locally by c_2 are transferred to a component responsible for their persistent storage. This storage component is not specified by EPCglobal and can be implemented arbitrarily, for example using relational or XML databases. The repository's Query Interface is exposed to external companies. Using Accessing Applications, companies such as c_1 and c_3 can use this interface. An Access Control mechanism is depicted as a logical component between the storage component and the Query Interface. Note that c_1 and c_3 might operate EPCIS Repositories and c_2 might employ an Accessing Application, too. This is not depicted for reasons of clarity.

As different business partners need different subsets of events to get their work done and companies usually try to disclose only a minimum amount of data, administrators need fine-grained means to define access rights. These access rights depend on the content of the events themselves and may also refer to data that is not stored yet, as new events are generated continuously. Instead of static lists defining which event can be accessed by whom, such as ACLs used in traditional AC systems, rules that are dynamically evaluated by the enforcement mechanism are required. Rules can refer to the content of the events and to contextual information such as business relationships and the current time. We introduce further requirements in Section 4.1.

3 Related Work

While security research in the context of RFID has mainly focused on privacy aspects, authentication, and secure reader-tag-communication [17, 11, 21], confidentiality of RFID-based data after it has been captured and stored has not received much attention so far [1]. At a first glance, sets of EPCIS Repositories can be considered distributed or federated databases, so respective access control models and mechanisms [6] seem to be viable starting points. However, the characteristics of the stored data and its volume pose new challenges [4, 23], especially regarding access control [14].

A recent NIST study [15] gives an assessment of established access control approaches. Popular models for non-military systems include Discretionary Access Control (DAC) (as implemented in the form of Access Control Lists (ACLs) [18] or Capabilities by popular operating systems), and Role-based Access Control [9]. The counterpart of DAC is formed by Non-Discretionary Access Control models (NDAC) such as Mandatory Access Control (MAC) whose best-known representative is the Bell-LaPadula model. Besides these established approaches, that do not form a strict taxonomy and can actually be combined in a number of ways, several application-specific security models and mechanisms have been developed. Temporal aspects have been addressed in TRBAC [3], which focuses on temporal availability and dependency of roles. Rule-Based Access Control (RuBAC) is a general term to describe AC systems that grant or deny access to resources based on rules defined by an administrator [15]. Research in the area of context-sensitive access control, e.g. [16, 12], strives to simplify access decisions by using environmental sensor data or other information not intrinsic to the system performing the AC. It is mainly focused on pervasive computing for personal use and has not been applied extensively to supply chain management before, even though approaches do exist [13].

Given the vast amount of data expected in future EPCIS Repositories [7, 23, 28], efficiency of the enforcement mechanism is an important issue. Since for a long time, relational databases will probably remain the dominant storage technology for such repositories, existing approaches to performing AC at the database level are relevant. Oracle's Virtual Private Databases [5] and Sybase's Row Level Security Model [25] use techniques similar to query rewriting [24],

basically adding predicates to each query to restrict the accessible rows and columns. Hippocratic Databases (HDB) [19] primarily aim at protecting patients' privacy in medical databases, but the concept can be applied to arbitrary relational databases. Instead of query rewriting, HDB replace the tables a query affects by prepared views. These views reflect the desired access policies in that protected rows, columns, and cells are removed compared to the original table. Our approach differs from all of the above in that we use an enhanced query rewriting technique that not only extends the original query, but recomposes it into several sub-queries in order to achieve flexible row, column, and cell based restrictions specified by rule-based policy definitions.

4 Efficient Access Control for EPCIS Repositories

In this section, we introduce a rule-based, content and context aware Policy Language for describing access rights to large amounts of RFID-based events. Based on specific requirements, the language design and its semantics are described. Furthermore, an enforcement mechanism based on query rewriting for relational databases is introduced.

4.1 Requirements

Based on a number of case studies [4, 23, 28] and previous work [14], we identified the following access control requirements:

Fine-grained disclosure control. Besides the ability to restrict access to certain events and event types, attribute-level restrictions need to be supported.

Content and context awareness. Access rights to events may depend on their respective content, as well as on contextual (external) information such as business relationships and temporal constructs.

Rules, range, and condition support. Because access rights are usually not assigned to individual events but to (continuously growing) sets of events, rules that may refer to ranges of events fulfilling certain conditions need to be supported.

Automatic reduction of result sets. If a user queries more information than he is authorized to access, the EPCIS repository has to return the respective allowed subset of events, instead of denying the whole query.

Query power restriction. To prevent information leakage due to inference, the query interface's flexibility needs to be restrictable per user or role.

Rapid execution. Due to the expected amount of events, efficiency of the enforcement mechanism in terms of memory consumption and execution time needs to be addressed.

4.2 AAL: A rule-based Policy Language for Auto-ID Events

Introductory Considerations Traditionally, each application or operating system employs proprietary policy representations. Recently, there is a trend

towards expressing policies using XML [29]. Besides their inherent extensibility, XML-based policy languages promise to be easier to read and edit and to offer better interoperability. With OASIS *XACML* [26, 20], an open industry standard for describing access policies using an XML-based language is available. Nonetheless, we decided to develop an own policy language for the following reasons. First, XACML's general purpose approach trades flexibility against simplicity. In our specific context, this clearly violates the principle of the "economy of mechanism" [22]. Second, despite its name, XACML not only specifies a language, but also an architecture how policies shall be evaluated and enforced. It recommends the separation of the AC mechanism by using a *Policy Decision Point* (PDP) and a *Policy Enforcement Point* (PEP). A PDP receives access requests, evaluates them using applicable policies and returns one of four predefined messages (*Permit*, *Deny*, *NotApplicable*, or *Indeterminate*). Based on such messages, a PEP permits or denies user's requests, issuing error messages if necessary. There are two problems with this architecture. Because in our scenario, access rights can be directly dependent on EPCIS Events' attributes, the PDP would have to access them in order to decide about requests. According to the XACML specification, a *Policy Information Point* (PIP) provides access to such external information. However, having to transfer large sets of events from the storage engine via a PIP to the PDP would introduce significant overhead. Furthermore, the property of XACML PDPs to only return one of four messages would make the "automatic reduction of result sets" (cmp. 4.1) impossible.

Language Design The main concept of our policy language called AAL (Auto-ID Authorization Language) is the notion of *Shares*. Each principal is assigned a set of Shares he is authorized to access. A Share defines a subset of EPCIS Events of a specific event type. Within a Share, all events disclosed to a user will contain only the attributes enumerated for this particular Share. The events of a Share are specified using a set of *Conditions*. Each Condition refers to exactly one attribute. All Conditions have to be fulfilled for an event to appear in a Share. Conditions are specified using *Values* that the respective attribute may hold. Values can be defined by enumeration (e.g. single EPCs) or using ranges and wildcards (such as EPC ranges and EPC patterns). These concepts can be formulated as follows:

$$Authorization(Principal) := Share_1 \cup \ldots \cup Share_k$$
$$Share(EventType, \{Attr_1, \ldots, Attr_l\}) := Condition_1 \cap \ldots \cap Condition_m$$
$$Condition(Attribute) := Value_1 \cup \ldots \cup Value_n$$

The following example defines two Shares. They are depicted as dark cells in Fig. 2. The table illustrates a subset of all ObjectEvents in a system, with each row representing an event (more descriptive attributes such as EPC and eventTime as well as content for all cells were omitted for reasons of clarity).

$$Share_1(ObjectEvent, \{a, b, e, f\}) = ((id \in \{1..4\}) \cap (a \in \{12..16\}))$$
$$Share_2(ObjectEvent, \{b, c, d, e\}) = ((id > 5) \cap (a < 20))$$

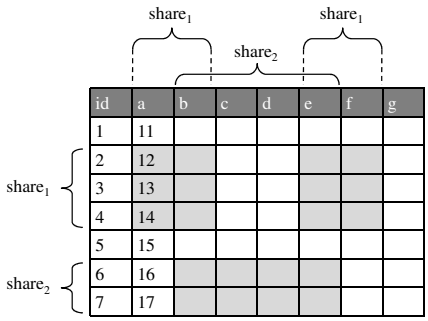

Fig. 2. Visualization of Shares

Language Semantics and Properties Our policy language is based on white-listing, i.e. everything that is not explicitly allowed is prohibited. This follows the principle of "Fail-safe defaults" [22]. While not preventing unwanted information leakage due to misconfigurations, this approach makes determining who is allowed to access which events much easier. If Shares containing different sets of attributes overlap (i.e. certain events are in more than one Share), the affected events will contain the union of the respective attributes. A policy can apply to any number of users or roles. Furthermore, policies can extend other policies additively. The structure and semantics of the policy language are illustrated in Fig. 3. Policy instances can be serialized in an XML dialect we defined using XML Schema.

```
Policy
   hasName name
   appliesTo (user|role)+
   extendsPolicies (policyName)*
   contains (Share
      refersTo eventType
      contains (visibleAttribute)+
      fullfillsAll (Condition
         refersTo eventAttribute|contextAttribute
         matchesAny (Value)+
      )*
   )+
```

Fig. 3. Structure and semantics of the policy language

Support for Contextual Information Using event's attributes such as EPC, eventTime, readPoint, or businessStep together with authorization rules allows for the specification of flexible policies. However, in certain situations contextual information extrinsic to the events is needed for authorative decisions. A built-in function to refer to the current time is `now()`. Using this function, a user can be granted temporally limited access to certain events by specifying *relative* time intervals. For example, he might be allowed to access all events from a certain business location that are not older than two days. Furthermore, a Condition may refer to a *contextAttribute*. Such attributes can be provided by external *Context Providers* and can be referenced in the form `contextProvider.contextAttribute`. For example, Context Providers can be used to retrieve the current user's identity and transactional information in order to base access control on business relationships.

4.3 An Efficient Enforcement Mechanism using Query Rewriting

Our enforcement mechanism is based on the assumption that EPCIS Events are stored inside relational databases. This assumption is valid because currently there is no alternative capable of inserting, storing and querying large amounts of structured data with acceptable performance.

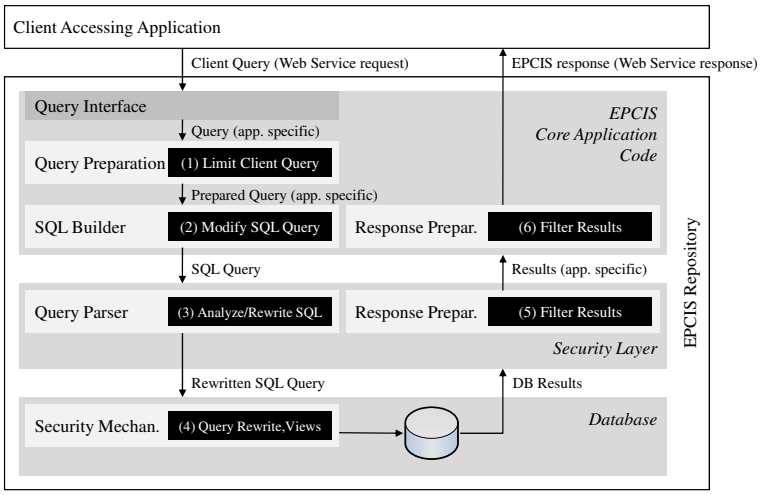

Fig. 4. Possible locations for AC mechanisms

Architectural Considerations Policy enforcement can be done at a number of logical locations.

First, the user's query submitted using the EPCIS Query Interface can be restricted to affect only the results he is allowed to access. Due to the limited

expressiveness of the EPCIS QI, this would not work in many cases. For example, the QI does not allow addressing multiple ranges of EPC at once.

Second, when an EPCIS Application builds an SQL query based on the user's request, it can enrich it with additional predicates that reflect the applicable Policy. The advantage of performing AC using modified SQL queries is that the performance and internal optimization mechanisms of existing databases can be leveraged.

Third, an EPCIS Application can construct an unrestricted SQL query, and instead of directly submitting it to the database driver, pass it through an additional security layer. This layer would analyze the query and rewrite it according to the respective policy. This has the advantage of encapsulating the security-relevant application code instead of mixing it into the query generation algorithm.

Fourth, built-in mechanisms of specific, proprietary database management systems can be used. We discussed some candidates in Section 3.

Fifth, a security layer can filter all result sets returned by the database before passing it on to the EPCIS application.

Sixth, this filtering can also be done by the EPCIS application itself.

Fig. 4 illustrates the six possibilities. Note that it does not depict an actual architecture. The black boxes represent potential locations for the AC mechanisms introduced above.

Translating Policies into Queries Our approach is based on rewriting an original SQL query to only refer to the events in the intersection of the set the user queried and the set he is authorized to access. As pointed out above, it can either be implemented inside the EPCIS, extending the SQL query building code, or in an external query analyzer/rewriter.

A Share definition can be translated into a relational algebra term as follows (a denotes attributes, v values):

$$Share_k = \pi_{a_1,\ldots,a_l}(\sigma_{(a_1=v_1 \vee \ldots \vee a_1=v_m) \wedge (\ldots) \wedge (a_n=v_o \vee \ldots \vee a_n=v_p)}(EventType))$$

Similarly, it can be expressed in SQL:

```
SELECT a_1,...,a_l FROM EventType WHERE
   (a_1=v_1 OR ... OR a_1=v_m) AND (a_n=v_o OR ... OR a_n=v_p)
```

The union of all Shares $Authorization = \bigcup_{k=1}^{n}(Share_k)$ defines the event subset a user is authorized to access. The intersection of $QueriedSet$ (the set he is trying to retrieve using his query) and $Authorization$ is the resulting event set to be returned: $ResultSet = Authorization \cap QueriedSet$. This means that both the Conditions and the attribute enumerations used to define $Authorization$ and $QueriedSet$ need to be intersected.

To build actual SQL statements, separate `SELECT` statements reflecting these intersections are constructed for each Share and combined using the `UNION` op-

erator. To keep the semantics of the events, the attributes also need to have the same order in each SELECT statements. Upon executing the query, the database will return the same events several times if they are contained in more than one share. They are aggregated into one event afterwards by the application. Fig. 5 illustrates the necessary steps using pseudocode.

```
query = "";
for each Share {
   Attributes = intersection(UserQuery.Attributes, Share.Attributes);
   Attributes = addNullColumns(Attributes);
   Conditions = removeUnauthorizedConditions
         (UserQuery.Conditions, Attributes);
   Conditions = insertContextValues(Conditions);
   Conditions = "((" + Conditions + ") AND (" + Share.Conditions + "))";
   if (!isFirst(Share)) { query += " UNION ALL "; }
   query += "SELECT " + Attributes + " FROM " + tableName
         + " WHERE " + Conditions;
}
resultSet = execute(query);
return filterAndMergeDuplicates(resultSet);
```

Fig. 5. Pseudocode for Query Construction

For example, assume the following user query and two Share definitions (also depicted in Fig. 6:

$$UserQuery_1(ObjectEvent, \{b, c, d\}) = ((id > 1) \cap (id < 7))$$
$$Share_1(ObjectEvent, \{a, b, e, f\}) = ((id \in \{1..4\}) \cap (a \in \{12..16\}))$$
$$Share_2(ObjectEvent, \{b, c, d, e\}) = ((id > 5) \cap (a < 20))$$

They will result in the following SQL statement:

```
SELECT b, NULL AS c, NULL AS d FROM ObjectEvent WHERE
   (id>1 AND id<7) AND ((id BETWEEN 1 AND 4)
   AND (a BETWEEN 12 AND 16))
UNION ALL
SELECT b, c, d FROM ObjectEvent WHERE
   (id>1 AND id<7) AND (id>5 AND a<20)
```

5 Evaluation

To evaluate our proposed policy language and mechanism, we implemented a prototype showcasing its applicability. Furthermore, we discuss how the requirements formulated in 4.1 are met and how the design principles for secure systems of Saltzer and Schroeder are reflected.

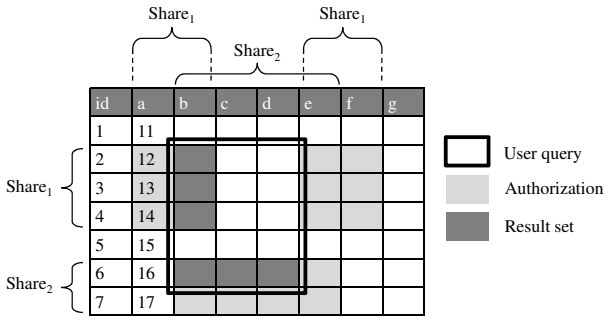

Fig. 6. Visualization of Shares and a user query

5.1 Practical Implementation

Our prototypical implementation is based on the *Accada RFID prototyping platform* [10] written in the Java programming language. The Accada EPCIS implementation runs on an *Apache Tomcat* 5.5 server, exposing its interfaces as Web Services based on the *Apache Axis* 1.4 library. For persistent storage, a *MySQL* 5.0 database and the *JDBC* library are employed. Accada's *Query Client* allows for querying the repository and displaying the returned events. We enhanced the Query Client's graphical interface by a user identity selection menu. Because authentication was out of the scope of our work, this replacement for a password or certificate mechanism is a viable simplification. For each user in the system, the policy definition is stored in a separate XML file named `<username>.pol`. To map the policy's XML structures to Java objects, the library *Simple*[4] 1.4 was used as a light-weighted alternative to *JAXB*. This way, extensions of the policy language do not entail significant updates of the program code. We modified the query generation code in Accada's `QueryOperationsModule.createEventQuery` method in the package `org.accada.epcis.repository`, constructing modified SQL queries based on the general idea presented in Section 4.3. Fig. 7 shows a screenshot of the Query Client, a policy definition and a part of a corresponding result set.

Our experience with the implementation of access control into Accada is twofold. First, it showed the technical feasibility of our approach. Reading XML-based policy files and transforming the rules into SQL to restrict a user's query therefore works in practice. Accada proved to be a solid basis for EPCglobal related prototyping activities. Second, the extension of Accada's complex query generation code turned out to be intricate. This is why architecturally, we consider placing the query rewriting code into a separate module a better approach (cmp. 4.3).

[4] http://simple.sourceforge.net/

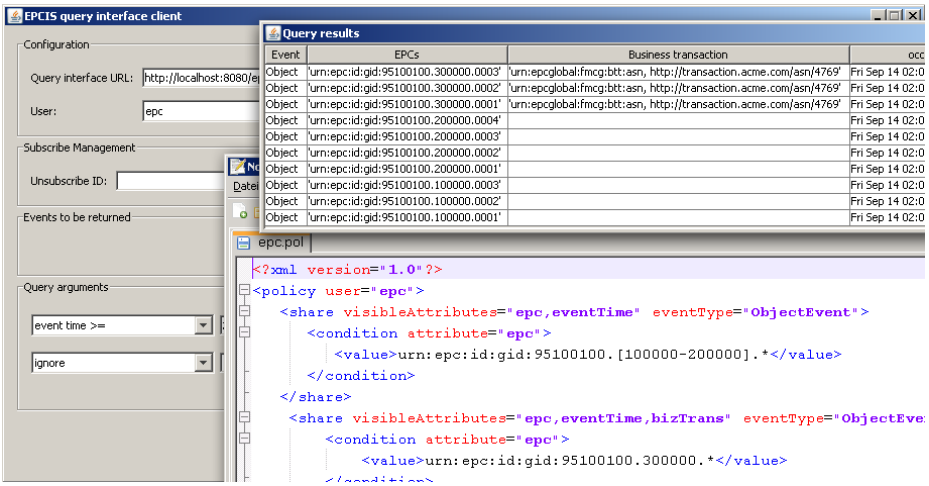

Fig. 7. Screenshot of Query Client, policy definition and result set

5.2 Addressing the Challenges

Our policy language AAL and the respective enforcement mechanism supports fine-grained Using a Share with one Condition, an individual event can be addressed (using suitable values for a simple or compound primary key). Because for such a Share, every column can be addressed directly by enumeration, cell-level control can be achieved.

Using our policy language, businesses can define context- and content-based rules who is allowed to access which information. Especially the ability to define relative time ranges and EPC patterns provides flexibility. At policy enforcement time, contextual attributes can be inserted, so information that is not known at policy design time can be included in the access decisions.

By intersecting the queried set and the authorized set, an automatic reduction of the result sets is performed. This is very important, because using an EPCIS compliant query interface, it is not possible for a user to tell the service that he is *not* interested in certain attributes. This, if an AC mechanism rejected all requests that refer to too many attributes, the user would not receive any information in case of a single violated server-side attribute restriction.

Our mechanism restricts the query interface's power by omitting user query restrictions that refer to attributes that are not accessible for him. This is done for each Share individually. However, this is just a basic restriction feature. To prevent undesired information leakage in the face of malicious queries, further investigations in the area of Inference Control would be needed.

Our query rewriting technique involves parsing the user query, resolving context attributes and constructing the final SQL statement. These steps are done once for each user request. The parsing of XML files only need to be performed once after a policy has changed. The overhead for constructing the modified

SQL query therefore only consists of several string operations. The actual access control is done by the database that executes the query. It can apply further query optimizations, for example removing or combining redundant predicates.

To further evaluate our approach, we discuss the fulfillment of the seven principles for secure systems of Saltzer and Schroeder [22]:

Economy of mechanism. Our design is simple, so this principle is fulfilled. The policy language's XML Schema definition comprises only 26 lines, compared to 380 lines of the XACML 2.0 policy schema.

Fail-safe defaults. We base access decisions on permission rather than exclusion, so this principle is fulfilled.

Complete mediation. We check every access for authority, so this principle is fulfilled.

Open design. Our design is not secret, so this principle is fulfilled.

Separation of privilege. We do not provide a two key mechanism, so this principle is not fulfilled.

Least privilege. The database runs at a higher security level than necessary in some situations, so this principle is not fulfilled.

Least common mechanism. This principle is not applicable.

Psychological acceptability. Since we do not provide a human interface, this principle is not applicable.

6 Conclusion and Future Work

Based on the access control requirements specific to Auto-ID based collaboration scenarios, we have presented a novel policy language and an efficient enforcement mechanism as well as their implementation. The policy language is expressed in XML and reflects the notion that companies will most probably prefer defining subsets of events to be shared using dynamic rules instead of static access control lists.

We have shown that an enforcement mechanism could be implemented either by restricting a user's query or by filtering a result set. In order to leverage relational databases' optimized query execution and to avoid high memory consumption, we argued that query rewriting is a viable approach. The techniques we presented to modify queries can also be applied to the generation of database views. These could increase query execution performance, possibly at the cost of storage space (in the case of materialized views). We discussed that most of our requirements can be met and that the system design reflects some "golden rules" for secure systems. By providing a prototypical implementation, we proved the plausibility of our approaches.

The outcome of our work are feasible means for administrators to restrict access to single EPCIS Repositories based on the known identities or roles of business partners. So far, this reflects the traditional paradigm of manually assigning rights to users of the system. However, in future dynamic supply chain scenarios, companies who do not know each other beforehand might need to

share certain data, with restrictions such as temporal constraints. This is especially true for *traceability queries*, which a certain stakeholder uses to determine all past locations and states of a given object or a class of objects. While we have shown how the enforcement of concrete access policies can be realized, we consider the management of such policies, including their generation, assignment, revokation, and maybe delegation, a challenging and open research issue. Considering the large amounts of both the information and the potential participants, overcoming the need to manually define every single access policy is highly desirable.

In our future work, we will target these issues as well as other access control challenges in global traceability networks such as access control for discovery services, inter-organizational role concepts and concepts for proving and delegating attributes and permissions.

References

1. Rakesh Agrawal, Alvin Cheung, Karin Kailing, and Stefan Schönauer. Towards Traceability across Sovereign, Distributed RFID Databases. In *IDEAS '06: Proceedings of the 10th International Database Engineering and Applications Symposium*, pages 174–184, Washington, DC, USA, 2006. IEEE Computer Society.
2. Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001.
3. Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. TRBAC: A Temporal Role-Based Access Control Model. In *RBAC '00: Proceedings of the fifth ACM workshop on Role-based access control*, pages 21–30, New York, NY, USA, 2000. ACM Press.
4. Christof Bornhövd, Tao Lin, Stephan Haller, and Joachim Schaper. Integrating Automatic Data Acquisition with Business Processes – Experiences with SAP's Auto-ID Infrastructure. In *Proceedings of the 30th International Conference on Very Large Data Bases*, pages 1182–1188, 2004.
5. Kristy Browder and Mary Ann Davidson. The Virtual Private Database in Oracle9iR2. Oracle Technical White Paper, Oracle Corporation, 500 Oracle Parkway, Redwood Shores, CA 94065, U.S.A., January 2002.
6. Sabrina De Capitani di Vimercati and Pierangela Samarati. Access Control in Federated Systems. In *NSPW '96: Proceedings of the 1996 workshop on New security paradigms*, pages 87–99, New York, NY, USA, 1996. ACM Press.
7. Hong-Hai Do, Jürgen Anke, and Gregor Hackenbroich. Architecture Evaluation for Distributed Auto-ID Systems. In *DEXA '06: Proceedings of the 17th International Conference on Database and Expert Systems Applications*, pages 30–34, Washington, DC, USA, 2006. IEEE Computer Society.
8. EPCglobal Inc. EPC Information Services (EPCIS) Version 1.0 Specification. http://www.epcglobalinc.org/standards/EPCglobal_EPCIS_Ratified_Standard_12April_2007_V1.0.pdf, April 2007.
9. D.F. Ferraiolo and D.R. Kuhn. Role-based access controls. In *15th National Computer Security Conference*, pages 554–563, Baltimore, MD, October 1992.
10. Christian Floerkemeier, Matthias Lampe, and Christof Roduner. Facilitating RFID Development with the Accada Prototyping Platform. In *Proceedings of PerWare Workshop 2007 at IEEE International Conference on Pervasive Computing and Communications*, New York, USA, March 2007.

11. Simson Garfinkel, Ari Juels, and Ravi Pappu. RFID Privacy: An Overview of Problems and Proposed Solutions. *IEEE Security and Privacy*, 3(3):34–43, May-June 2005.
12. Christin Groba, Stephan Groß, and Thomas Springer. Context-Dependent Access Control for Contextual Information. In *ARES '07: Proceedings of the The Second International Conference on Availability, Reliability and Security*, pages 155–161, Washington, DC, USA, 2007. IEEE Computer Society.
13. Eberhard Grummt and Ralf Ackermann. Proof of Possession: Using RFID for large-scale Authorization Management. In Max Mühlhäuser, Alois Ferscha, and Erwin Aitenbichler, editors, *Constructing Ambient Intelligence: AmI-07 Workshops Proceedings*, LNCS. Springer-Verlag Berlin Heidelberg, 2008.
14. Eberhard Grummt, Markus Müller, and Ralf Ackermann. Access Control: Challenges and Approaches in the Internet of Things. In *Proceedings of the IADIS International Conference WWW/Internet 2007*, volume 2, pages 89–93, Vila Real, Portugal, October 2007.
15. Vincent C. Hu, David F. Ferraiolo, and D. Rick Kuhn. Assessment of Access Control Systems. Interagency Report 7316, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930, September 2006.
16. R. J. Hulsebosch, A. H. Salden, M. S. Bargh, P. W. G. Ebben, and J. Reitsma. Context sensitive access control. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 111–119, New York, NY, USA, 2005. ACM Press.
17. Ari Juels. RFID Security and Privacy: A Research Survey. *IEEE Journal on Selected Areas in Communication*, 24(2):381–394, February 2006.
18. Butler Lampson. Protection. In *Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems*, pages 437–443, Princeton University, 1971.
19. K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. DeWitt. Limiting Disclosure in Hippocratic Databases. In *Proceedings of the 30th International Conference on Very Large Data Bases*, pages 108–119, Toronto, Canada, August 2004.
20. Markus Lorch, Seth Proctor, Rebekah Lepro, Dennis Kafura, and Sumit Shah. First Experiences Using XACML for Access Control in Distributed Systems. In *XMLSEC '03: Proceedings of the 2003 ACM workshop on XML security*, pages 25–37, New York, NY, USA, 2003. ACM Press.
21. Pedro Peris-Lopez, Julio Cesar Hernandez-Castro, Juan Estevez-Tapiador, and Arturo Ribagorda. RFID Systems: A Survey on Security Threats and Proposed Solutions. In *11th IFIP International Conference on Personal Wireless Communications – PWC'06*, volume 4217 of *Lecture Notes in Computer Science*, pages 159–170. Springer-Verlag, September 2006.
22. J. H. Saltzer and M. D. Schroeder. The Protection of Information in Computer Systems. In *Proceedings of the IEEE*, volume 63, pages 1278–1308, September 1975.
23. Sanjay Sarma. Integrating RFID. *ACM Queue*, 2:50–57, 2004.
24. Michael Stonebraker and Eugene Wong. Access control in a relational data base management system by query modification. In *ACM 74: Proceedings of the 1974 annual conference*, pages 180–186, New York, NY, USA, 1974. ACM Press.
25. Sybase, Inc. New Security Features in Sybase Adaptive Server Enterprise. Technical Whitepaper, 2003.

26. Tim Moses (Editor). eXtensible Access Control Markup Language (XACML) Version 2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, February 2005.
27. Ken Traub, Greg Allgair, Henri Barthel, Leo Burstein, John Garrett, Bernie Hogan, Bryan Rodrigues, Sanjay Sarma, Johannes Schmidt, Chuck Schramek, Roger Stewart, and KK Suen. The EPCglobal Architecture Framework – EPCglobal Final Version of 1 July 2005. http://www.epcglobalinc.org/standards/Final-epcglobal-arch-20050701.pdf, July 2005.
28. Fusheng Wang and Peiya Liu. Temporal Management of RFID data. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 1128–1139. VLDB Endowment, 2005.
29. Mariemma I. Yagüe. Survey on XML-Based Policy Languages for Open Environments. *Journal of Information Assurance and Security*, 1:11–20, 2006.