

ACADA:

Access Control-driven Architecture with Dynamic Adaptation

Óscar Mortágua Pereira, Rui L. Aguiar

Instituto de Telecomunicações
DETI, University of Aveiro
Aveiro, Portugal
{omp,ruilaa}@ua.pt

Maribel Yasmina Santos

Centro Algoritmi
University of Minho
Guimarães, Portugal
maribel@dsi.uminho.pt

Abstract— Programmers of relational database applications use software solutions (Hibernate, JDBC, LINQ, ADO.NET) to ease the development process of business tiers. These software solutions were not devised to address access control policies, much less for evolving access control policies, in spite of their unavoidable relevance. Currently, access control policies, whenever implemented, are enforced by independent components leading to a separation between policies and their enforcement. This paper proposes a new approach based on an architectural model referred to here as the Access Control-driven Architecture with Dynamic Adaptation (ACADA). Solutions based on ACADA are automatically built to statically enforce access control policies based on schemas of Create, Read, Update and Delete (CRUD) expressions. Then, CRUD expressions are dynamically deployed at runtime driven by established access control policies. Any update in the policies is followed by an adaptation process to keep access control mechanisms aligned with the policies to be enforced. A proof of concept based on Java and Java Database Connectivity (JDBC) is also presented.

Keywords— access control; software architecture; adaptive systems.

I. INTRODUCTION

Software systems have increasingly played a key role in all dimensions of our existence as humans, such as transport operators, financial movements, e-health, e-governance and national/international security. They are responsible for managing sensitive data that needs to be kept secure from unauthorized usage. Access control policies (ACP) are a critical aspect of security. ACP are aimed at preventing unauthorized access to sensitive data and is usually implemented in a three phase approach [1]: security policy definition; security model to be followed; and, finally, security enforcement mechanism. Security policies define rules through which access control is governed. The four main strategies for regulating access control policies are [2, 3]: discretionary access control (DAC), mandatory access control (MAC), Role-based access control (RBAC) and credential-based access control. Security models provide formal representations [4-8] for security policies. Security enforcement mechanisms implement the security policy formalized by the security model. ACP exist to keep sensitive data safe, mostly kept and managed by database management systems. Among the several paradigms, the

relational database management systems (RDBMS) continue to be one of the most successful one to manage data and, therefore, to build database applications. Beyond RDBMS, software architects use other current software solutions (CuSS), such as JDBC [9], ODBC [10], JPA [11], LINQ [12], Hibernate [13], ADO.NET [14] and Ruby on Rails [15] to ease the development process of business tiers of database applications. Unfortunately, CuSS were devised to tackle the impedance mismatch issue [16], leaving ACP out of their scope. Current mechanisms to enforce ACP to data residing in a RDBMS consist of designing a separate security layer, following one of two different approaches: traditional and PEP-PDP:

1) The traditional approach is based on a security software layer developed by security experts using RDBMS tools. ACP architecture vary from RDBMS to RDBMS but comprise several entities, such as users, roles, database schemas and permissions. They are directly managed by RDBMS and are completely transparent to applications. Their presence is only noticed if some unauthorized access is detected. Basically, before being executed, SQL statements are evaluated by RDBMS to check their compliance with the established ACP. If any violation is detected, SQL statements are rejected, otherwise they are executed.

2) The PEP-PDP approach consists in a security software layer with two main functionalities: the policy decision point (PDP) and the policy enforcement point (PEP), as defined in XACML [17] and used in [18], see Figure 1. The PEP intercepts users requests for accessing a resource protected by an ACP (Figure 1, 1) and enforces the decision to be performed by PDP on this access authorization. PDP evaluates requests to access a resource against the ACP to decide whether to grant or to deny the access (Figure 1, 2). If authorization is granted, the action is sent by the PEP to RDBMS to be executed (Figure 1, 3) and, if no other

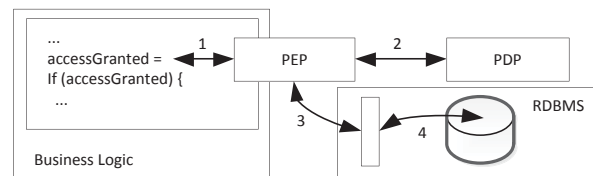


Figure 1. PEP-PDP approach.

access control exists, the action is executed by the RDBMS (Figure 1, 4). PDPs are designed and configured by security experts and exist as independent components. PEPs are intentionally inserted in key points of the source code to enforce PDP decisions.

Both approaches impose a sharp separation between ACP and the mechanisms responsible for their enforcement. This fragility is also apparent in CuSS: security layers exist to enforce ACP and CuSS exist to ease the development process of database applications. This separation of roles entails four important drawbacks regarding the usage of current CuSS:

1) For programmers who use CuSS, this separation demands a complete mastering on the established ACP and on their dependency on database schemas. This mastering is very difficult to be sustained when the complexity of ACP increases, usually coupled by an increase in complexity of databases schemas.

2) Programmers who use CuSS are free to write any CRUD expression opening a security gap. CRUD expressions may be in compliance with the formalized ACP but violating security rules that are not possible to be formalized. ACP are suited to control the access to schema objects but not to control the information SQL statements may get from them. If a user has the permission to, for example, read a set of columns from one or more tables, it is not possible to prevent any SQL statement from reading that data. Select statements may select raw data or may use aggregate functions, for example, to select critical statistical information, opening a possible security gap.

3) Whenever ACP are updated, the correspondent access control mechanisms have to be updated in advance. There is no way to automatically translate ACP into access control mechanisms of CuSS.

4) Some ACP need to be hard-coded to manage runtime constraints. There is no way to automatically update these scattered and hidden hard-coded access control mechanisms in current CuSS.

To tackle the aforementioned drawbacks we propose a new architecture for CuSS, herein referred to as Access Control-driven Architecture with Dynamic Adaptation (ACADA). Software solutions cease to be of general use and become specialized solutions to address specific business areas, such as accountability, warehouse and customers. They are automatically built from a business architectural model, enforcing ACP defined by a security expert. ACP are statically enforced by typed objects driven by schemas of Create, Read, Update and Delete (CRUD) expressions. Then, CRUD expressions are deployed at runtime in accordance with the established ACP. Any modification in the ACP is followed by an adaptation process to keep access control mechanisms aligned with the policies to be enforced. A proof of concept based on Java, JDBC [9] and SQL Server 2008 is also presented.

This paper is organized as follows: section II presents the motivation and related word; section III presents the

proposed approach; section IV presents a proof of concept and, finally, section V presents the final conclusion.

II. MOTIVATION AND RELATED WORK

CuSS have been devised to improve the development process of business logic mainly for tackling the impedance mismatch [16]. From them, two categories have had a wide acceptance in the academic and commercial forums: 1) Object-to-Relational mapping (O/RM) tools [19, 20] (LINQ [12], Hibernate [13], Java Persistent API (JPA) [11], Oracle TopLink [21], CRUD on Rails [15]) and 2) Low Level API (JDBC [9], ODBC [10], ADO.NET [14]). Other solutions, such as embedded SQL [22] (SQLJ [23]), have achieved some acceptance in the past. Others were proposed but without any general known acceptance: Safe Query Objects [24] and SQL DOM [25].

Listing 1 shows the usage of four CuSS (JDBC, ADO.NET, JPA and LINQ) for updating the attribute *totalValue* returned by the query “*select clientId, ttlValue from Orders where date=2012-01-31*”. Programmers are completely free to edit any CRUD expression (CRUD expressions are encoded inside strings), to execute it (line 2, 9, 20, 27) and to update the attribute *ttlValue* (line 4-5, 14-16, 21-24, 28-29). There is no sign of any ACP: neither for the CRUD expression being executed nor for the updated attribute. Beyond updating *totalValue*, nothing prevents programmers from writing source code to update any other attribute. Programmers have no guidance either on the established ACP or on the underlying database schema. Only after writing and running the source code, programmers become aware of any ACP violation or any database schema nonconformity. Moreover, this same source code may be

```

1 // JDBC - Java
2 rs=st.executeQuery(sql);
3 rs.next();
4 rs.updateFloat("ttlValue", newValue);
5 rs.updateRow();
6
7 //ADO.NET - C#
8 SqlDataAdapter da=new SqlDataAdapter();
9 da.SelectCommand=new SqlCommand(sql,conn);
10 SqlCommandBuilder cb=new SqlCommandBuilder(da);
11 DataSet ds=new DataSet();
12 da.Fill(ds,"Orders");
13 DataRow dr=ds.Tables["Orders"].Rows[0];
14 dr["ttlValue"]=totalValue;
15 cb.GetUpdateCommand();
16 da.Update(ds,"Orders");
17
18 // JPA - Java
19 Query qry=em.createNamedQuery(sql,Orders.class);
20 Orders o=(Orders)qry.getSingleResult();
21 em.getTransaction().begin();
22 o.setTtlValue(value);
23 em.persist(o);
24 em.getTransaction().commit();
25
26 //LINQ - C#
27 Order ord=(from o in Orders select o).Single();
28 ord.ttlValue=value;
29 db.SubmitChanges();

```

Listing 1. Examples using CuSS

```

select o.clientId,SUM(o.ttlValue) as ttlValue
from Orders as o
where o.date between '2012-01-01' and '2012-01-31'
group by o.clientId
order by o.clientId desc

```

Listing 2. CRUD expression with aggregate function.

used to execute an infinite number of different CRUD expressions requiring the same ACP, such as the one shown in Listing 2. There is no way to avoid this type of security violation. Even if a PEP was used, it would not solve any of the hurdles previously presented. An example of the need for evolving ACP, is the designation of a secretary Susanne to be temporally allowed to update clients' *ttlValue*. Her role has to be changed but roles of other secretaries are to be kept unchanged. ACP foresee this possibility by using the delegation concept. The problem is the lack of preparedness of CuSS to accommodate this situation. Source-code needs to be modified to accommodate the new permission for Susanne. The situation will further deteriorate if the permission is to be only granted while she is within the facilities of the company. The use of hard-coded mechanisms to enforce ACP entails maintenance activities on source-code of client-side components of database applications whenever ACP evolve. CuSS and current access control mechanisms are not prepared to seamlessly accommodate and enforce these evolving ACP.

To address these issues several solutions have been proposed.

SELINKS [18] is a programming language in the type of LINQ and Ruby on Rails which extends Links [26]. Security policies are coded as user-defined functions on DBMS. Through a type system named as Fable, it is assured that sensitive data is never accessed directly without first consulting the appropriate policy enforcement function. Policy functions, running in a remote server, check at runtime what type of actions users are granted to perform, basically controlling more efficiently what RDBMS are currently able to do, and this way not tackling the need to master ACP and database schemas. Moreover, if ACP evolve there will be no way to automatically accommodate the modifications in the client-side components.

Jif [27] extends Java with support for information access control and also for information flow control. The access control is assured by adding labels that express ACP. Jif addresses some relevant aspects such as the enforcement of security policies at compile time and at runtime. Anyway, at development time programmers will only be aware of inconsistencies after running the Jif compiler. In spite of its valuable contribution, Jif does not address the announced goals of this research.

Rizvi et al. [28] uses views to filter contents of tables and simultaneously to infer and check at runtime the appropriate authorization to execute any query. The process is transparent for users and queries are rejected if they do not have the appropriate authorization. This approach has some disadvantages: 1) the inference rules are complex and time consuming; 2) security enforcement is transparent, so users do not know that their queries are run against views; 3)

programmers cannot statically check the correctness of queries which means they are not aware of either the ACP or the underlying database schema.

Morin et al. [29] uses a security-driven model-based dynamic adaptation process to address simultaneously access control and software evolution. The approach begins by composing security meta-models (to describe access control policies) and architecture meta-models (to describe the application architecture). They also show how to map (statically and dynamically) security concepts into architectural concepts. This approach is mainly based on establishing bindings between components from different layers to enforce security policies. Authors didn't address the key issue of how to statically incorporate the established security policies in software artifacts.

Differential-privacy [30] has had significant attention from the research community. It is mainly focused on preserving privacy from statistical databases. It really does not directly address the point here under discussion. The interesting aspect is Frank McSherry's [31] approach to address differential-privacy: PINQ - a LINQ extension. The key aspect is that the privacy guarantees are provided by PINQ itself not requiring any expertise to enforce privacy policies. PINQ provides the integrated declarative language (SQL like, from LINQ) and simultaneously provides native support for differential-privacy for the queries being written.

III. ACADA: PROPOSED APPROACH

In this section a new architecture, ACADA, is proposed for CuSS. We first introduce an overview for the proposed approach. Then we introduce some relevant aspects of CuSS from which ACADA will evolve. Then, CRUD Schemas are presented as key entities of ACADA. Finally, ACADA is presented.

A. Overview

ACADA is an architecture for software solutions used in business tiers of database applications. Each software solution derived from ACADA, herein known as Access Control-driven Component with Dynamic Adaptation

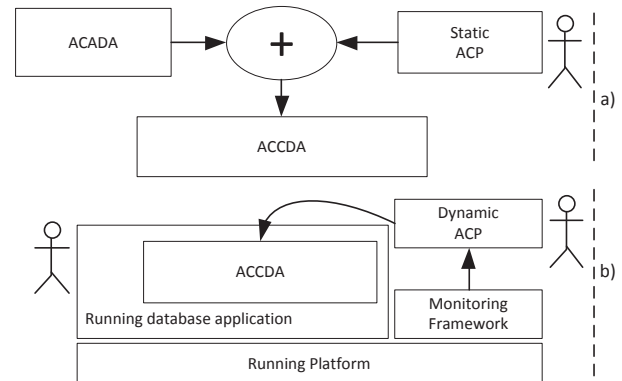


Figure 2. Proposed approach for the adaptation of ACCDA a) static composition and b) dynamic adaptation.

(ACCD A), is customized to address a specific need of a business area, such as accountability, warehouse and sales. Then, at runtime, they are dynamically adapted to be kept aligned with the established ACP. This approach combines a static composition of ACCDA and a dynamic adaptation to the running context as shown in Figure 2. During the static composition, Figure 2 a), static parts of ACP are used to build an ACCDA based on an architectural model (ACADA). Static parts of ACP comprise the information needed to build the business logic to manage CRUD expressions. Any modification in the static parts compels to a new static composition. During the dynamic adaptation, see Figure 2 b), CRUD expressions are dynamically assigned and unassigned to running ACCDA in accordance with ACP defined for each user. This process is continuous and may have as input data from a monitoring framework and from security experts. Security experts modify ACP, for example, to allow secretary Susanne to update clients' *ttlValue* and, therefore, to use the necessary business logic and necessary CRUD expressions. Monitoring framework updates ACP, for example, only to allow Susanne to update clients' *ttlValue* while she is within the facilities of the company.

B. Relevant Aspects of CuSS

To proceed with a more detailed presentation it is advisable to learn and understand CuSS and the context in which CRUD expressions are executed. Identifying a common base for current approaches is a key aspect to devise ACADA. The following shared functionalities are emphasized:

1) To promote reusability of CRUD expressions, parameters may be used to define runtime values. Parameters are mainly used to define runtime values for clause conditions and for column lists. Listing 3 shows an Update CRUD expression with four parameters: *a*, *b* and *c* are columns and *d* is a condition.

```
update table set a=?, b=?, c=? where d=?
```

Listing 3. CRUD expression with parameters.

2) If CRUD expression type is *Insert*, *Update* or *Delete*, its execution returns a value indicating the number of affected rows.

3) Data returned by CRUD expressions of type *Select* is managed by a local memory structure (LMS) internally created by CuSS. Some LMS are readable only and others are readable and modifiable. Modifiable LMS provide additional functionalities to modify their internal content: update data, delete data and insert new data. These actions, are equivalent to CRUD expressions and the results are committed into the host RDBMS.

Thus, CRUD expressions are used at two levels: at the application level and at the LMS level. At the application level CRUD expressions are explicitly used, while at the LMS level CRUD expressions are implicitly used. In both cases, to guarantee compliance with established ACP and with database schemas, CRUD expressions need to be

emanated from the established ACP and from database schemas and not from programmers' will. In reality, CRUD expressions and LMS are the key assets of CuSS to interact with RDBMS. They are the entities used to read data from databases and to alter the state of databases.

C. Crud Schemas

CRUD expressions and LMS are two key entities of ACADA. They are the entities used to interact with databases and, therefore, the privileged entities through which ACP may be enforced. To this end, ACADA formalizes CRUD expressions and LMS using a schema herein known as CRUD schema. A CRUD schema is a set of services needed to manage the execution of CRUD expressions and the associated LMS (only for Select CRUD expressions). It comprises four independent parts: a) a mandatory type schema – the CRUD type - query (Select) or execute (Insert, Update or Delete); b) an optional parameter schema – to set the runtime values for the conditions used inside SQL clauses, such as the “where” and “having” clauses and runtime values for column lists (only for Insert and Update CRUD expressions); c) mandatory result schema for Insert, Update and Delete CRUD expressions – to handle the number of affected rows during the CRUD expressions execution and, finally, d) a mandatory LMS schema for Select CRUD expressions – to manage the permissions on the LMS.

Table I shows a possible definition for the permissions on an LMS derived from the CRUD expression *Select a,b,c,d,e from table*. This access matrix [32] like representation, defines for each attribute of this LMS, which LMS functionalities (read, update, insert, delete) are authorized. *delete* action is authorized in a tuple basis and, therefore, it is executed as an atomic action for all attributes.

TABLE I. TABLE OF PERMISSIONS IN A LMS

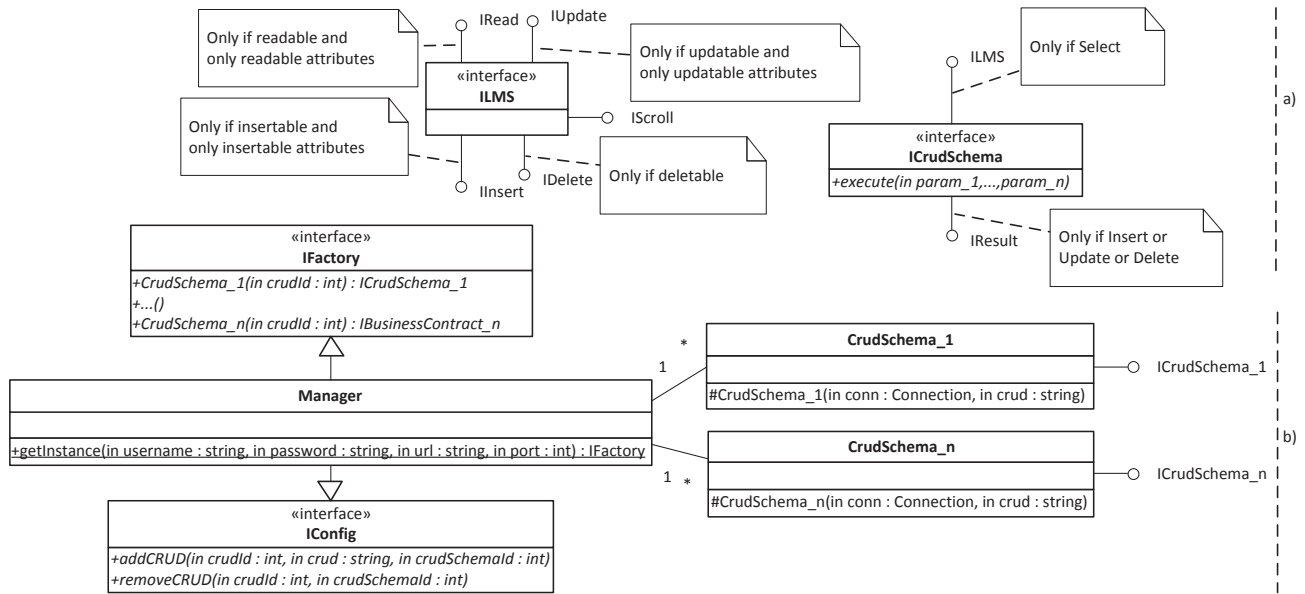
	a	b	c	d
Read	yes	no	yes	yes
Update	no	yes	no	yes
Insert	yes	yes	no	no
delete	yes			

D. ACADA Presentation

Figure 3 presents a class diagram for an ACADA model, for building ACCDA. Figure 3 a) presents the entities used to define CRUD schemas. Figure 3 b) presents the final class diagram of ACADA. There are six types of entities: ILMS, ICrudSchema, Manager, IFactory, IConfig and CrudSchema:

ILMS (used for Select CRUD expressions only) defines the permissions on LMS, following the approach presented in Table I: IRead defines the readable attributes, IUpdate defines the updatable attributes, IInsert defines the insertable attributes and IDelete defines if LMS's rows are deletable. Additionally, ILMS also uses IScroll to define the scrollable methods to be made available.

ICrudSchema is used to model the business logic for each



CRUD schema instance – see *ICrudSchema_1*, ..., *ICrudSchema_n*, in Figure 3 b). It comprises a mandatory method, *execute(param_1,...,param_n)*, to set the parameter schema and to execute CRUD expressions, and two optional interfaces: *ILMS* and *IResult*. *IResult* (for Insert, Update and Delete CRUD expressions only) implements the result schema.

CrudSchema is used to implement *ICrudSchema*. The arguments *conn* and *crud* are a connection object to a database and the CRUD expression to be managed, respectively. Each *CrudSchema* is able to manage any CRUD expression with equivalent schema. CRUD expressions with the same CRUD Schema are herein known as sibling CRUD expressions. Listing 4 presents two simple sibling CRUD expressions: both are Select, both have the same select list, none has column list or condition list parameters. This property is an opportunity to extend the adaptation capability of ACADA. In practice each *CrudSchema* is able to manage an infinite number of sibling CRUD expressions. Thus, any *CrudSchema* used by UserA is able to manage not one CRUD expression but one set of sibling CRUD expressions and the same *CrudSchema* may be used by UserB to manage a different set of sibling CRUD expressions.

```
Select * from table;
Select * from table where id=10;
```

Listing 4. 2 Sibling CRUD expressions.

Manager implements two interfaces (*IFactory* and *IConfig*) and is the entry point for creating instances of ACCDA (using *getInstance*, authentication is required). *url* and *port* are used to connect to a component responsible for the dynamic adaptation process and for the authentication of users.

IConfig is used to dynamically adapt running instances of ACCDA to users previously authenticated. The dynamic process comprises the deployment of CRUD expressions and also the required information to set the connection to RDBMS (not shown). Each CRUD expression is assigned to a *CrudSchema* responsible for its management. *IConfig* is implemented using a socket to decouple ACCDA from components responsible for managing the dynamic adaptation process.

IFactory is used to create instances of *CrudSchema*. Users request the access to a *CrudSchema* and to a CRUD expression. The access is granted or denied depending on the ACP defined by the dynamic adaptation process.

IV. PROOF OF CONCEPT

In this section a proof of concept based on Java and JDBC (sqljdbc4) for SQL Server 2008, is presented. A component, herein known as ACEngine, was developed to automatically create releases of ACCDA. The biggest challenge was centered on the approach to be followed to formalize CRUD schemas to be used to define the target business area. Several approaches were considered, among them XML and standard Java interfaces. In spite of being less expressive than XML, Java interfaces proved to be an efficient and effective approach. Programmers do not need to use a different development environment, interfaces are basic entities of any object-oriented programming language and are widely used, interfaces are easily edited and maintained and, finally, CRUD schemas have also been defined as interfaces, see Figure 3. These were the fundamental reasons for having opted for Java interfaces in detriment of XML.

ACEngine accepts as input, for each CRUD schema, one interface extending all the necessary interfaces as defined in

ICrudSchema and shown in Figure 3. Then, through reflection, AEngine detects which interfaces are defined and which methods need to be implemented to automatically create the source code.

A component for the dynamic adaptation process was also created. The main information is organized around users. For each user it is defined its authentication parameters (username and password), the assigned CRUD expressions and the correspondent CrudSchemas. Any modification in this information is immediately sent to users running ACCDA instances.

The example to be presented is based on the Select and on the permissions used in Table I.

Figure 4 shows the four interfaces used to formalize the LMS's permissions, which are in agreement with Table I. CuSS use the same access methods for updating and for inserting attributes. This approach prevents the separation between update permissions and insert permissions. Therefore, to overcome this limitation, access methods of IUpdate and IInsert have been given different names. IUpdate use a prefix *u* and IInsert use a prefix *i*. Some additional methods, such as *uUpdate()* and *iBeginInsert()* are used to implement the update and insert protocols defined by JDBC for LMS.

```
public interface IRead {
    int rA() throws SQLException;
    int rC() throws SQLException;
    int rD() throws SQLException;
}

public interface IUpdate {
    void uBeginUpdate() throws SQLException;
    int uB(int value) throws SQLException;
    int uD(int value) throws SQLException;
    void uUpdate() throws SQLException;
    void uCancelUpdate() throws SQLException;
}

public interface IInsert {
    void iBeginInsert() throws SQLException;
    int iA(int value) throws SQLException;
    int iB(int value) throws SQLException;
    void iInsert() throws SQLException;
    void iCancelInsert() throws SQLException;
}

public interface IDelete {
    void dDelete() throws SQLException;
}
```

Figure 4. IRead, IUpdate, IInsert and IDelete interfaces.

Figure 5 presents the usage of ACCDA from a programmer's perspective. An attempt is done to create a new ACCDA instance (line 29). It will raise an exception if a connection to ACDynam fails or if authentication fails. Authentication is processed by ACDynam and if it succeeds, ACDynam transfers to ACCDA all the CRUD expressions, in accordance with the ACP assigned to the authenticated user. Then, an attempt is made to create an instance of a crudSchema for managing the CRUD expression identified by token 1 (line 30). As previously mentioned, programmers cannot edit CRUD expressions. They are only allowed to use CRUD expressions made available by ACDynam, overcoming this way the security gap of CuSS. If it fails (user is not authorized to

```
27 private void use(String un,String pw,String url,int port){
28     try {
29         IFactory f=Manager.getInstance(un,pw,url,port);
30         ICrudSchema s=f.crudSchema(1);
31         s.execute();
32         while (s.moveToNext()) {
33             a=s.rA();
34             c=s.rC();
35             d=s.rD();
36             s.uBeginUpdate();
37             s.u
38         } catch (uB(int value) int
39         } catch (uBeginUpdate() void
40         // . uCancelUpdate() void
41         } catch (uD(int value) int
42         // . uUpdate() void
43     }
44 }
```

Figure 5. ACCDA from the programmer's perspective.

execute the CRUD expression), an exception is raised. If not, CRUD expression is executed (line 31) and LMS is scrolled row by row (line 32). The dynamic adaptation is on behalf of ACDynam that, at any time, may modify the permission to use this CRUD expression. There is no need to update any source-code this way overcoming CuSS to be adapted to evolving ACP. The three readable attributes are read (line 33-35). Update protocol is started (line 36). Auto-completion window (line 38-43) shows the available methods to update attributes of LMS, relieving programmers from mastering the established ACP and database schema. This type of guided-assistance is available for all operations involving ACCDA, this way overcoming the need for mastering ACP and database schemas when using CuSS.

V. CONCLUSION

In this paper a new architecture (ACADA) was presented to devise solutions driven by ACP and able to be dynamically adapted to deal with evolving ACP. The adaptation process of each ACCDA release is achieved in a two phase approach: static composition and dynamic adaptation. Static composition is triggered whenever a maintenance activity is necessary in CRUD Schemas. Dynamic adaptation is a continuous process where CRUD expressions are deployed to running ACCDA instances in accordance with ACP. ACP are dynamically updated by a monitoring framework and/or by security experts.

Source code is automatically generated from an architectural model and from ACP defined by a security expert. In opposite to CuSS, programmers using ACCDA are relieved from mastering ACP and database schemas, and also from writing CRUD expressions. Security is ensured by preventing programmers from writing CRUD expressions and by controlling dynamically, at runtime, the set of CRUD expressions that each user may use. Evolving ACP are seamlessly supported and enforced by ACCDA. An independent and external component keeps the access control mechanisms of ACCDA updated at runtime by assigning and unassigning CRUD expressions. This adaptation capability of ACCDA avoids maintenance activities in the core client-side components of database applications when ACP evolve. The adaptation capacity is significantly improved by

CrudSchema design which, theoretically, is able to manage an infinite number of sibling CRUD expressions.

Summarizing, ACADA overcomes the four drawbacks of CuSS. This achievement is mostly grounded on its two phase adaptation process: static composition and dynamic adaptation.

It is expected that this work may open new perspectives for enforcing evolving ACP in business tier components of database applications.

REFERENCES

- [1] P. Samarati and S. D. C. d. Vimercati, "Access Control: Policies, Models, and Mechanisms," *Foundations of Security Analysis and Design*, pp. 137-109, 2001.
- [2] P. Samarati and S. D. C. d. Vimercati, "Access Control: Policies, Models, and Mechanisms," presented at the Revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design on Foundations of Security Analysis and Design: Tutorial Lectures, 2001.
- [3] S. D. C. d. Vimercati, S. Foresti, and P. Samarati, "Recent Advances in Access Control - Handbook of Database Security," M. Gertz and S. Jajodia, Eds., ed: Springer US, 2008, pp. 1-26.
- [4] D. Basin, J. Doser, and T. Lodderstedt, "Model Driven Security: From UML Models to Access Control Infrastructures," *ACM Transactions on Software Engineering and Methodology*, vol. 15, pp. 39-91, 2006.
- [5] R. Breu, G. Popp, and M. Alam, "Model Based Development of Access Policies," *International Journal on Software Tools for Technology Transfer*, vol. 9, pp. 457-470, 2007.
- [6] T. Doan, S. Demurjian, T. C. Ting, and A. Ketterl, "MAC and UML for Secure Software Design," presented at the Proceedings of the 2004 ACM Workshop on Formal Methods in Security engineering, Washington DC, USA, 2004.
- [7] I. Ray, N. Li, R. France, and D.-K. Kim, "Using UML to Visualize Role-based Access Control Constraints," presented at the Proceedings of the ninth ACM Symposium on Access Control Models and Technologies, Yorktown Heights, New York, USA, 2004.
- [8] OASIS, "eXtensible Access Control Markup Language (XACML)," ed: OASIS Standard.
- [9] M. Parsian, *JDBC Recipes: A Problem-Solution Approach*. NY, USA: Apress, 2005.
- [10] Microsoft. (2011 Oct). Microsoft Open Database Connectivity. Available: [http://msdn.microsoft.com/en-us/library/ms710252\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms710252(VS.85).aspx)
- [11] D. Yang, *Java Persistence with JPA*: Outskirts Press, 2010.
- [12] M. Erik, B. Brian, and B. Gavin, "LINQ: Reconciling Object, Relations and XML in the .NET framework," in *ACM SIGMOD International Conference on Management of Data*, Chicago, IL, USA, 2006, pp. 706-706.
- [13] B. Christian and K. Gavin, *Hibernate in Action*: Manning Publications Co., 2004.
- [14] G. Mead and A. Boehm, *ADO.NET 4 Database Programming with C# 2010*. USA: Mike Murach & Associates, Inc., 2011.
- [15] D. Vohra, "CRUD on Rails - Ruby on Rails for PHP and Java Developers," ed: Springer Berlin Heidelberg, 2007, pp. 71-106.
- [16] M. David, "Representing database programs as objects," in *Advances in Database Programming Languages*, F. Bancilhon and P. Buneman, Eds., ed N.Y.: ACM, 1990, pp. 377-386.
- [17] OASIS. (2012 Feb). XACML - eXtensible Access Control Markup Language. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
- [18] B. J. Corcoran, N. Swamy, and M. Hicks, "Cross-tier, Label-based Security Enforcement for Web Applications," presented at the Proceedings of the 35th SIGMOD International Conference on Management of Data, Providence, Rhode Island, USA, 2009.
- [19] W. Keller, "Mapping Objects to Tables - A Pattern Language," in *European Conference on Pattern Languages of Programming Conference (EuroPLoP)*, Irsee, Germany, 1997.
- [20] R. Lammel and E. Meijer, "Mappings Make data Processing Go 'Round: An Inter-paradigmatic Mapping Tutorial," in *Generative and Transformation Techniques in Software Engineering*, Braga, Portugal, 2006.
- [21] Oracle. (2011 Oct). Oracle TopLink. Available: <http://www.oracle.com/technetwork/middleware/toplink/overview/index.html>
- [22] J. W. Moore, "The ANSI binding of SQL to ADA," *Ada Letters*, vol. XI, pp. 47-61, 1991.
- [23] Part 1: SQL Routines using the Java (TM) Programming Language, 1999.
- [24] R. C. William and R. Siddhartha, "Safe query objects: statically typed objects as remotely executable queries," in *27th International Conference on Software Engineering*, St. Louis, MO, USA, 2005, pp. 97-106.
- [25] A. M. Russell and H. K. Ingolf, "SQL DOM: compile time checking of dynamic SQL statements," in *27th International Conference on Software Engineering*, St. Louis, MO, USA, 2005, pp. 88-96.
- [26] E. Cooper, S. Lindley, P. Wadler, and J. Yallop, "Links: Web Programming Without Tiers," presented at the Proceedings of the 5th International Conference on Formal Methods for Components and Objects, Amsterdam, The Netherlands, 2007.
- [27] D. Zhang, O. Arden, K. Vikram, S. Chong, and A. Myers. (2011 Dec). Jif: Java + information flow. Available: <http://www.cs.cornell.edu/jif/>
- [28] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy, "Extending Query Rewriting Techniques for Fine-grained Access Control," presented at the Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, Paris, France, 2004.
- [29] B. Morin, T. Mouelhi, F. Fleurey, Y. L. Traon, O. Barais, and J.-M. Jézéquel, "Security-Driven Model-based Dynamic Adaptation," presented at the Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, Antwerp, Belgium, 2010.
- [30] C. Dwork, "Differential Privacy: A Survey of Results," presented at the Proceedings of the 5th International Conference on Theory and Applications of Models of Computation, Xi'an, China, 2008.
- [31] F. McSherry, "Privacy Integrated Queries: An Extensible Platform for Privacy-preserving Data Analysis," *Commun. ACM*, vol. 53, pp. 89-97, 2010.
- [32] B. W. Lampson, "Protection," *SIGOPS Operating Systems Review*, vol. 8, pp. 18-24, 1974.