

**Óscar Narciso  
Mortágua Pereira**

**DACA: Arquitetura para Implementação de  
Mecanismos Dinâmicos de Controlo  
de Acesso em Camadas de Negócio**

**DACA: Architecture to Implement Dynamic  
Access Control Mechanisms  
on Business Tier Components**



universidade de aveiro



Universidade do Minho

**U. PORTO**

Programa de Doutoramento em Informática  
das Universidades do Minho, Aveiro e Porto





**Óscar Narciso  
Mortágua Pereira**

**DACA: Arquitetura para Implementação de  
Mecanismos Dinâmicos de Controlo de Acesso em  
Camadas de Negócio.**

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Ciências da Computação (MAP-i), realizada sob a orientação científica do Prof. Doutor Rui L. Aguiar, Professor Associado com Agregação, do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro em co-orientação com a Prof. Doutora Maribel Yasmina Santos, Professora Associada com Agregação, do Departamento de Sistemas de Informação da Universidade do Minho.



Dedico este trabalho:

- ao meu pai, à minha mãe
- à minha família: Lygia, Lia e Nuno



o júri

**presidente**

Prof. Doutor Carlos Alberto Diogo Soares Borrego  
Professor Catedrático da Universidade de Aveiro

Prof. Doutor Arnaldo Carvalho Martins  
Professor Catedrático da Universidade de Aveiro

Prof. Doutor Marco Paulo Amorim Vieira  
Professor Auxiliar da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Prof. Doutor João Costa Seco  
Professor Auxiliar da Faculdade de Ciências e Tecnologia da Universidade de Lisboa

Prof. Doutor Rui Luís Andrade Aguiar (orientador)  
Professor Associado com Agregação da Universidade de Aveiro

Prof. Doutora Maribel Yasmina Campos Alves Santos (co-orientadora)  
Professora Associada com Agregação da Escola de Engenharia da Universidade do Minho.





## **agradecimentos**

O modo como se desenvolveu e se concluiu este trabalho deve muito ao apoio científico prestado e também às condições criadas e proporcionadas pelo Prof. Doutor Rui Luís Aguiar. Sem a sua colaboração e compreensão, o resultado seria inevitavelmente diferente. A ele o meu sincero obrigado.

Realço a disponibilidade total, constante e pronta da Prof. Doutora Maribel Yasmina Santos na colaboração científica que prestou. A ela o meu sincero obrigado.

Finalmente, realço a contribuição de todos os elementos do grupo ATNoG que de alguma forma também colaboraram para o sucesso deste meu trabalho. Não posso deixar de destacar o Prof. Doutor. Diogo Gomes, o Prof. Doutor João Paulo Barraca e o técnico André Rainho. A todos também o meu sincero obrigado.



## palavras-chave

**Controlo de acesso, componentes, arquitecturas de software, sistemas adaptativos, base de dados relacionais, camadas de negócio.**

## resumo

Controlo de acesso é um desafio para a engenharia de software nas aplicações de bases de dados. Atualmente, não há uma solução satisfatória para a implementação dinâmica de mecanismos finos e evolutivos de controlo de acesso (FGACM) ao nível das camadas de negócio de aplicações de bases de dados relacionais. Para solucionar esta lacuna, propomos uma arquitetura, aqui referida como Arquitetura Dinâmica de Controlo de Acesso (DACA). DACA permite que FGACM sejam dinamicamente construídos e atualizados em tempo de execução de acordo com as políticas finas de controlo de acesso (FGACP) estabelecidas. DACA explora e utiliza as características das Call Level Interfaces (CLI) para implementar FGACM ao nível das camadas de negócio. De entre as características das CLI, destacamos o seu desempenho e os diversos modos para acesso a dados armazenados em bases de dados relacionais. Na DACA, os diversos modos de acesso das CLI são envolvidos por objetos tipados derivados de FGACM, que são construídos e atualizados em tempo de execução. Os programadores prescindem dos modos tradicionais de acesso das CLI e passam a utilizar os dinamicamente construídos e atualizados. DACA compreende três componentes principais: Policy Server (repositório de meta-data dos FGACM), Dynamic Access Control Component (componente da camada de negócio que é responsável pela implementação dos FGACM) e Policy Manager (*broker* entre DACC e Policy Server). Ao contrário das soluções atuais, DACA não é dependente de qualquer modelo de controlo de acesso ou de qualquer política de controlo de acesso, promovendo assim a sua aplicabilidade a muitas e diversificadas situações. Com o intuito de validar DACA, foi concebida e desenvolvida uma solução baseada em Java, Java Database Connectivity (JDBC) e SQL Server. Foram efetuadas duas avaliações. A primeira avalia DACA quanto à sua capacidade para dinamicamente, em tempo de execução, implementar e atualizar FGACM e, a segunda, avalia o desempenho de DACA contra uma solução sem FGACM que utiliza o JDBC normalizado. Os resultados recolhidos mostram que DACA é uma solução válida para implementar FGACM evolutivos em camadas de negócio baseadas em CLI.



**keywords**

**Access control, business tiers, software architecture, components, adaptive systems, relational databases, business tiers.**

**abstract**

Access control is a software engineering challenge in database applications. Currently, there is no satisfactory solution to dynamically implement evolving fine-grained access control mechanisms (FGACM) on business tiers of relational database applications. To tackle this access control gap, we propose an architecture, herein referred to as Dynamic Access Control Architecture (DACA). DACA allows FGACM to be dynamically built and updated at runtime in accordance with the established fine-grained access control policies (FGACP). DACA explores and makes use of Call Level Interfaces (CLI) features to implement FGACM on business tiers. Among the features, we emphasize their performance and their multiple access modes to data residing on relational databases. The different access modes of CLI are wrapped by typed objects driven by FGACM, which are built and updated at runtime. Programmers prescind of traditional access modes of CLI and start using the ones dynamically implemented and updated. DACA comprises three main components: Policy Server (repository of metadata for FGACM), Dynamic Access Control Component (DACC) (business tier component responsible for implementing FGACM) and Policy Manager (broker between DACC and Policy Server). Unlike current approaches, DACA is not dependent on any particular access control model or on any access control policy, this way promoting its applicability to a wide range of different situations. In order to validate DACA, a solution based on Java, Java Database Connectivity (JDBC) and SQL Server was devised and implemented. Two evaluations were carried out. The first one evaluates DACA capability to implement and update FGACM dynamically, at runtime, and, the second one assesses DACA performance against a standard use of JDBC without any FGACM. The collected results show that DACA is an effective approach for implementing evolving FGACM on business tiers based on Call Level Interfaces, in this case JDBC.



## TABLE OF CONTENTS

1	Introduction.....	1
1.1	Problem Definition .....	1
1.2	Solution Proposal.....	4
1.3	Research questions .....	5
1.4	Contributions .....	6
1.5	Computational Tools and Infrastructure .....	8
1.6	Thesis Organization.....	8
2	Background and State of the Art .....	9
2.1	Basic Access Control Concepts .....	9
2.1.1	Access Control Strategies.....	10
2.1.2	Architectures for Access Control Mechanisms .....	12
2.1.3	Dimensions of Access Control Mechanisms .....	16
2.2	Current tools for Building Business Tiers .....	19
2.2.1	O/RM tools and ADO.NET.....	19
2.2.2	Call Level Interfaces .....	21
2.2.3	Other proposals .....	27
2.3	JDBC .....	29
2.3.1	JDBC Overview .....	29
2.3.2	JDBC Approach to Call Level Interfaces Functionalities .....	30
2.3.3	JDBC Class Diagram .....	33
2.4	Current Approaches to Implement Access Control .....	36
2.4.1	Current Techniques .....	36
2.4.2	Related Work .....	41
2.5	Summary .....	52
3	From Call Level Interfaces Towards the DACA.....	53
3.1	Concepts .....	53
3.1.1	CRUD Schema .....	53
3.1.2	Business Schema.....	57
3.1.3	Business Entity.....	58
3.2	Modelization of Call Level Interfaces .....	59
3.2.1	Motivation.....	59
3.2.2	Proposed Approach for the Modelization of CLI.....	62
3.3	Componentization of CLI.....	66
3.3.1	Components.....	66
3.3.2	Adaptation Process.....	67
3.4	Access Control.....	72
3.5	Summary .....	72
4	DACA: Dynamic Access Control Architecture .....	75
4.1	Fine-grained Access Control Mechanisms .....	75
4.2	General Architecture.....	76
4.2.1	Phases of the DACA .....	76
4.2.2	General Operation of the DACA.....	79
4.3	The DACA Components .....	80
4.3.1	The DACC .....	80
4.3.2	Policy Server .....	88
4.3.3	Policy Manager .....	89
4.4	Summary .....	90
5	Proof of Concept.....	91
5.1	The DACA Platform.....	91
5.1.1	Scenario.....	92
5.1.2	Awareness of FGACM.....	94
5.1.3	Security Configurator.....	96
5.1.4	Security Keeper.....	98

5.1.5	DbProof.....	99
5.2	Performance Assessment.....	100
5.2.1	Methodology.....	101
5.2.2	Collected Results.....	105
5.3	Results Evaluation.....	109
5.3.1	Dynamic FGACM on business tiers.....	109
5.3.2	Security.....	110
5.3.3	FGACM awareness.....	110
5.3.4	Preservation of CLI Advantages.....	110
5.4	Summary.....	111
6	Conclusion.....	113
6.1	Overview.....	113
6.2	Contributions.....	114
6.3	Discussion.....	114
6.4	Future Work.....	117
6.4.1	Extending DACC to Support Additional Access Modes.....	117
6.4.2	Fine-grained Access Control Policies for the DACA.....	117
6.4.3	Concurrent Approach of Call Level Interfaces.....	117
6.4.4	Multi-function Components.....	118
6.4.5	Extending FGACP to the Runtime Values of CRUD expressions.....	118
6.4.6	Orchestration of Business Entities.....	119
6.4.7	The DACA Based on LINQ.....	119
	References.....	121
	Annex A – Logical model for metadata of FGACM.....	131
	Annex B - Concurrency on CLI.....	135
B.1	CTSA- The Wrapper Approach.....	135
B.1.1	CTSA Presentation.....	135
B.1.3	Proof of Concept.....	138
B.1.4	CTSA Performance Assessment.....	140
B.1.5	Conclusion.....	146
B.2	Embedded Approach.....	146
B.2.1	Presentation.....	146
B.2.2	Architecture.....	147
B.2.2.1	Individual Cache.....	147
B.2.2.2	Shared Cache.....	147
B.2.3	Performance Assessment.....	148
B.3	Conclusion.....	148
	Annex C – ABTC: Multi-purpose Adaptable Business Tier Components.....	149
C.1	Introduction.....	149
C.2	ABTC.....	149
C.2.1	Adaptation Process.....	149
C.2.2	Architecture Presentation.....	151
C.3	Proof of Concept.....	153
C.4	Discussion.....	154
C.5	Conclusion.....	155

## TABLE OF FIGURES

Figure 1. Typical usage of CLI (JDBC).....	2
Figure 2. Simplified block diagram of the DACA.....	4



Figure 3. Centralized access control mechanism. ....	13
Figure 4. Mixed architecture based on PEP and PDP. ....	16
Figure 5. Example based on ADO.NET. ....	20
Figure 6. Example based on JPA. ....	20
Figure 7. Example based on LINQ. ....	20
Figure 8. LMS with 5 tuples (rows) and 6 attributes (a till f). ....	24
Figure 9. CLI and DACA access mechanisms. ....	26
Figure 10. Types of JDBC drivers and their dependency on other components. ....	29
Figure 11. Declaration of variables. ....	30
Figure 12. Use of forward-only and read-only statement. ....	31
Figure 13. Use of forward-only and read-only prepared statement. ....	31
Figure 14. Use of scrollable and updatable statement. ....	31
Figure 15. Insert a row using a prepared statement. ....	32
Figure 16. Examples of transaction with JDBC. ....	32
Figure 17. Methods to scroll on LMS. ....	32
Figure 18. JDBC class diagram. ....	33
Figure 19. Connection interface. ....	33
Figure 20. Statement interface. ....	34
Figure 21. PreparedStatement interface. ....	34
Figure 22. ResultSet interface. ....	35
Figure 23. Enforcement of RBAC in Java EE. ....	51
Figure 24. Enforcement of RBAC in ORBAC. ....	51
Figure 25. Three CRUD expressions with different combinations of CRUD Schemas. ....	55
Figure 26. Two sibling CRUD expressions. ....	55
Figure 27. Partial example of how to implement the permissions of Table 3 on LMS. ....	58
Figure 28. Typical JDBC/CLI drawbacks. ....	61
Figure 29. Business Schema for the modelization of CLI: CRUD-Model. ....	63
Figure 30. Block diagram for the modelization process of CLI. ....	64
Figure 31. Partial view of a Business Entity based on the CRUD-Model. ....	64
Figure 32. Example shown in Figure 28 but based on the CRUD-Model. ....	65
Figure 33. Block diagram for the static approach: a) service composition and b) service allocation. ....	69
Figure 34. Block diagram for the dynamic service composition. ....	69
Figure 35. Attributes shared by all CRUD expressions. ....	71
Figure 36. Example of one Multiple Business Schema implementation. ....	71
Figure 37. General architecture of the DACA. ....	77
Figure 38. Concept of permission in the DACA. ....	78
Figure 39. Simplified block diagram of DACC. ....	80
Figure 40. Class diagram of DACC. ....	81
Figure 41. Business Entity class diagram. ....	86
Figure 42. ILMS class diagram for LMS. ....	86
Figure 43. Access control Meta-model. ....	89
Figure 44. Block diagram for the proof of concept. ....	92
Figure 45. Hierarchy of roles. ....	93
Figure 46. Role_B2 definition. ....	96
Figure 47. Programmers awareness about FGACM for Role_B2. ....	96
Figure 48. Application definition. ....	97
Figure 49. Role_B2 definition. ....	97
Figure 50. Business Schema IPrd_s definition. ....	97

Figure 51. IRead definition. .... 97

Figure 52. Definition of all CRUD expressions. .... 98

Figure 53. Security keeper. .... 98

Figure 54. DbProof. .... 99

Figure 55. Business Schemas implemented for user User\_A. .... 99

Figure 56. LMS interfaces for Cat\_s Business Schema. .... 100

Figure 57. Graphics for scenario SS<sub>r</sub>. .... 106

Figure 58. Graphics for scenarios SS<sub>i</sub>, SS<sub>u</sub> and SS<sub>d</sub>. .... 107

Figure 59. Graphic for scenarios SI, SU and SD. .... 108

Figure 60. Wrapping approach to provide the *getMet*. .... 111

Figure 61. DbProof implemented inADO.NET. .... 116

Figure 62. Logical model for *the proof of concept*. .... 131

Figure 63. CTSA main protocols. .... 137

Figure 64. CTSA class diagram. .... 138

Figure 65. CTSA constructor. .... 138

Figure 66. Partial view of IRead protocol. .... 139

Figure 67. Partial view of IScroll protocol. .... 139

Figure 68. Set and store the execution context. .... 140

Figure 69. CTSA from users's perspective. .... 140

Figure 70. Std\_Student schema. .... 141

Figure 71.  $E_{(c-jdbc,p,s)} / E_{(c-ctsa,p,s)}$  chart. .... 144

Figure 72.  $E_{(c-jdbc,p,s)} / E_{(c-ctsa,p,s)}$  details. .... 144

Figure 73.  $E_{(c-jdbc,p,u)} / E_{(c-ctsa,p,u)}$  chart. .... 145

Figure 74.  $E_{(c-jdbc,p,i)} / E_{(c-ctsa,p,i)}$  chart. .... 145

Figure 75. Implemented and tested scenarios. .... 150

Figure 76. Class diagram of ABTC. .... 152

## TABLE OF TABLES

Table 1. Access matrix to a table with attributes *a, b, c* and *d*. .... 10

Table 2. Main protocols of LMS. .... 26

Table 3. Example of a table of permissions in a LMS (Indirect Access Mode). .... 58

Table 4. Roles and the correspondent permissions for the implemented scenario. .... 94

Table 5. Strategy to collect and compute measurements. .... 101

Table 6. Collected measurements for a) TBS, TBW and for b) RAM in ns. .... 102

Table 7. Scenarios for the Select expression: algorithms and typical component usage. .... 103

Table 8. Scenarios for the Insert, Update and Delete expressions: algorithms and typical component usage. .... 104

Table 9. Exclusive access mode approaches. .... 136

Table 10. Algorithm for  $E_{(c-jdbc,p,\gamma)}$  assessment. .... 142

Table 11. Algorithm for  $E_{(c-jdbc,p,\gamma)}$  assessment. .... 143

Table 12. CRUD expressions and Business Schemas for the implemented scenarios. .... 154

## TABLE OF LISTINGS

Listing 1. Definition of table <i>user</i> in Ur/Web. ....	47
Listing 2. Policy definition in Ur/Web. ....	47
Listing 3. Policy definition in $\lambda_{DB}$ . ....	48
Listing 4. Query rewritten in T-SQL. ....	49
Listing 5. Four examples of CRUD expressions. ....	57

## ACRONYMS

<b>ABAC</b>	Attribute-Based Access Control
<b>ABTC</b>	Adaptable Business Tier Component
<b>ABTC_Dynamic</b>	Adaptable Business Tier Component - Dynamic
<b>ABTC_Static</b>	Adaptable Business Tier Component - Static
<b>ACP</b>	Access Control Policy
<b>API</b>	Application Programming Interface
<b>BE</b>	Business Entity
<b>BS</b>	Business Schema
<b>BW</b>	Business Worker
<b>CBAC</b>	Credential Based Access Control
<b>CLI</b>	Call Level Interfaces
<b>CRUD</b>	Create Read Update Delete
<b>CRUD-Model</b>	CRUD-Model
<b>CTSA</b>	Concurrent Tuple Set Architecture
<b>C-CTSA</b>	Component – Concurrent Tuple Set Architecture
<b>C-JDBC</b>	Concurrent – Java Database Connectivity
<b>DAC</b>	Discretionary Access Control
<b>DACA</b>	Dynamic Access Control Architecture
<b>DACC</b>	Dynamic Access Control Component
<b>DAM</b>	Direct Access Mode
<b>DCA</b>	Denial Category Assignment
<b>DDL</b>	Data Definition Language
<b>DFMAC</b>	Dynamic Fine-grained Meta-level Access Control
<b>DRBAC</b>	Dynamic Role Based Access Control
<b>FGAC</b>	Fine-grained Access Control
<b>FGACM</b>	Fine-grained Access Control Mechanism
<b>FGACP</b>	Fine-grained Access Control Policy
<b>JDBC</b>	Java Database Connectivity
<b>GUI</b>	Guided User Interface
<b>IAM</b>	Indirect Access Mode
<b>IDE</b>	Integrated Development Environment
<b>IIS</b>	Internet Information Server
<b>JIF</b>	Java + Information Flow
<b>LMS</b>	Local Memory Structure
<b>MAC</b>	Mandatory Access Control
<b>MDE</b>	Model Driven Engineering

<b>ODBC</b>	Open Database Connectivity
<b>O/RM</b>	Object-o-Relational Model
<b>PAP</b>	Policy Administration Point
<b>PCA</b>	Permission Category Assignment
<b>PDP</b>	Policy Decision Point
<b>PEP</b>	Policy Enforcement Point
<b>PEP-PDP</b>	PEP PDP
<b>PIP</b>	Policy Information Point
<b>RBAC</b>	Role Based Access Control
<b>RDBAC</b>	Reflective Database Access Control
<b>RDBMS</b>	Relational Database Management System
<b>SAC</b>	Semantic Access Control
<b>SQL</b>	Structured Query Language
<b>TA<sub>i</sub></b>	Time to execute a method with 10 arguments and returning void
<b>TBS</b>	Time to instantiate a Business Session
<b>TBW</b>	Time to instantiate a Business Worker
<b>TR<sub>i</sub></b>	Time to execute a method with no arguments
<b>XACML</b>	Extended Access Control Markup Language

# 1 Introduction

Fine-grained access control (FGAC) is a critical security issue in many software systems, mainly when policies evolve over time. Software systems are increasingly involved in all dimensions of our existence as humans. When operating in critical organizations, such as airports, hospitals, banks and power plants, they need to be available 24 hours a day and always operating under a high level of security. They manage data from which all day decisions are taken, many of them critical. To prevent any security violation, several security measures are taken such as user authentication, data encryption and secure connections. Another relevant security concern is access control [Samarati, '01b; Vimercati, '08], which *“is concerned with limiting the activity of legitimate users.”* [Sandhu, '94]. Basically, access control is a process to supervise every request to access a protected resource, in our case data residing inside relational database management systems (RDBMS), by determining whether the permission should be granted or denied. While access control is enforced at the table level, fine-grained access control (FGAC) is enforced at the column and row level. Currently, there isn't any known solution to automatically build and keep updated, at runtime, fine-grained access control mechanisms (FGACM) on business tiers of relational database applications and in accordance with the established fine-grained access control policies (FGACP).

This chapter is organized as follows. Section 1.1 describes the problem being addressed. Section 1.2 briefly presents a solution to overcome the identified problem. Section 1.3 states the research questions to be addressed in this thesis. Section 1.4 enumerates and describes the contributions of this thesis. Section 1.5 presents the tools and infra-structured used during the thesis development process and, finally, section 1.6 presents the thesis organization.

## 1.1 Problem Definition

Critical data are mostly kept and managed by database management systems. Among the several paradigms, the relational paradigm continues to be one of the most successful to manage data and, therefore, to build database applications. To be useful, data need to be stored, updated and retrieved from databases. To this end, software architects use software tools to ease the development process of business tiers. Two groups of software tools are widely accepted in commercial and academic forums: O/RM tools (Java Persistent API [Yang, '10], LINQ [Erik, '06], Hibernate [Christian, '04] and Ruby on Rails [Vohra, '07]) and Call Level Interfaces (CLI) [ISO, '03] (JDBC [Parsian, '05], ODBC [Microsoft, '92], ADO.NET [Mead, '11]). Unfortunately, none of these tools addresses access control, much less when the policies evolve over time. These tools were mainly devised and designed to tackle the impedance mismatch issue [David, '90].

We now leverage the importance of CLI as the perfect choice for building business tiers whenever performance is considered a key requirement [Cook, '05] in detriment of other requirements such as productivity, usability and maintainability. Some of the features that contribute positively for a high performance of CLI are:

#### Fine tune control

CLI are low level API (Application Programming Interface) that provide programmers with a fine tune control to manage and optimize the environment in which Create, Read, Update and Delete (CRUD) expressions are executed.

#### Use of the native SQL language

Native SQL statements are encoded inside strings, this way keeping the performance and the full expressiveness of the SQL language.

#### Multi-access mode to data

CLI support several modes to access to data residing on relational databases. In each situation programmers are free to choose the access mode that better addresses their needs. Among the several access modes the use of the native SQL language is the most well-known. Chapter 2 thoroughly describes the access modes used by the DACA.

```
46 | sql="Select * from Products p Where p.productId=?";
47 | ps=conn.prepareStatement(sql,
48 |     ResultSet.TYPE_FORWARD_ONLY,
49 |     ResultSet.CONCUR_UPDATABLE);
50 | rs=ps.executeQuery();
51 | if (rs.next()) {
52 |     switch(op) {
53 |         case read:
54 |             productName=rs.getString("productName");
55 |             // ... read more attributes
56 |             break;
57 |         case update:
58 |             rs.updateString("productName",productName);
59 |             // ... update more attributes
60 |             rs.updateRow();
61 |             break;
62 |         case insert:
63 |             rs.moveToInsertRow();
64 |             rs.updateString("productName",productName);
65 |             // ... insert more attributes
66 |             rs.insertRow();
67 |             break;
68 |         case delete:
69 |             rs.deleteRow();
70 |             break;
71 |     }
72 | }
```

Figure 1. Typical usage of CLI (JDBC).

In spite of these key advantages, CLI (as other software tools) do not address access control. Figure 1 presents a typical usage of CLI in this case based on JDBC. From this example we see that there is no sign of any FGACP or any FGACM. Programmers are free to write any CRUD expression encoded inside strings (Figure 1: line 46) and execute them (Figure 1: line 47-50). The presented CRUD expression is a Select expression and, therefore, it returns a relation. CLI provide protocols from which programmers read the retrieved data (Figure 1: line 54-55), update the returned data (Figure 1: line 58-60), insert new data (Figure 1: line 63-66) and delete returned data (Figure 1: line 69). There is no possibility to prevent programmers from writing this type of source-code and therefore there is no possibility to guide programmers to write source code in accordance with any established FGACP.

To overcome this lack of access control of current software tools, several approaches are proposed by the commercial and the academic communities. Security experts, instead of exploiting the advantages of CLI to implement FGACM, build additional security layers specially crafted to control the access to protected data. These security layers are responsible for evaluating authorization to perform actions on database objects and also to execute them if permission is granted. These security layers convey several drawbacks, among them four are emphasized:

#### Awareness gap about the policies and about the mechanisms

Programmers of business tiers and application tiers are expected to master the established access control policies at development time. This mastering process is very difficult to be sustained when the complexity of access control policies increases. Programmers do only get aware of any security violation after having written the source code. Awareness of any violation may take place at compile time but in most of the cases it is only obtained at runtime. Before compiling the source or running the database application, there is no possibility to statically validate the authorized actions during the development process of business tiers.

#### Security gap

The SQL language is characterized by its endless expressiveness capacity and programmers of business tiers and application tiers are not restricted to write any CRUD expression. This freedom opens the possibility for the existence of security gaps. Current techniques can hardly guarantee safety for all CRUD expressions [Shi, '09; Wang, '07]. Even if safety is guaranteed, programmers are always before additional techniques with increased complexity to express the policies to be enforced [Caires, '11; Chaudhuri, '07; Chlipala, '10; Corcoran, '09; Fischer, '09; Gary, '07; Hicks, '10; LeFevre, '04; Rizvi, '04; Wang, '07; Yang, '12]. Additional techniques and, above all, increased complexity frequently leads to the possibility of opening unwanted security gaps.

#### Wastage of CLI features

Architecture of current security layers is not based on architecture of CLI, this way preventing the use of their advantages by business and application tiers.

#### Maintenance activities

Currently, any modification in the policies implying modifications in the access control mechanisms forces a maintenance activity on the security layers and/or on the business logic to be carried out in advance. Currently, there is no way to translate access control policies automatically into access control mechanisms and/or into the business logic. Maintenance is

very often a critical activity when modifications are necessary at the client-side of database applications with a great amount and distant equipment.

## 1.2 Solution Proposal

To tackle the aforementioned drawbacks of current security layers, we propose a new architecture herein referred to as Dynamic Access Control Architecture (DACA). Security layers relying on the DACA are dynamically built at their instantiation time on the client-side of database applications and are continuously updated to enforce any modification in the established access control policies. The dynamic adaptation of business tiers may leverage security systems based on models@run.time [Blair, '09] to continuously keep security layers aligned with the policies they must enforce. To take advantage of CLI features, the implemented access control mechanisms are closely aligned with the architecture of CLI and with the services they provide. Among the services provided by CLI, the DACA makes use of two access modes as it will be described in chapter 4. To overcome possible security gaps of current solutions, users are restricted to use only the permissions provided by the FGACM. Among the restrictions, users can only use CRUD expressions made available by the implemented FGACM. Finally, the DACA provides programmers of business tiers with a complete awareness about the implemented FGACM this way relieving them from mastering FGACM while writing source code.

To unveil the proposed architecture, Figure 2 presents a simplified block diagram, though incomplete, of the DACA. The overall operation is as follows:

- A security layer (business tier relying on CLI – it is a client-side layer) is dynamically built and kept updated (from an architectural model based on CLI and from the policies kept in a server) to implement access control mechanisms in accordance with the established policies. The security layer is composed by typed objects driven by FGACM;
- Client applications access database objects through the security layer;
- Security layer uses standard CLI to interact with database objects.

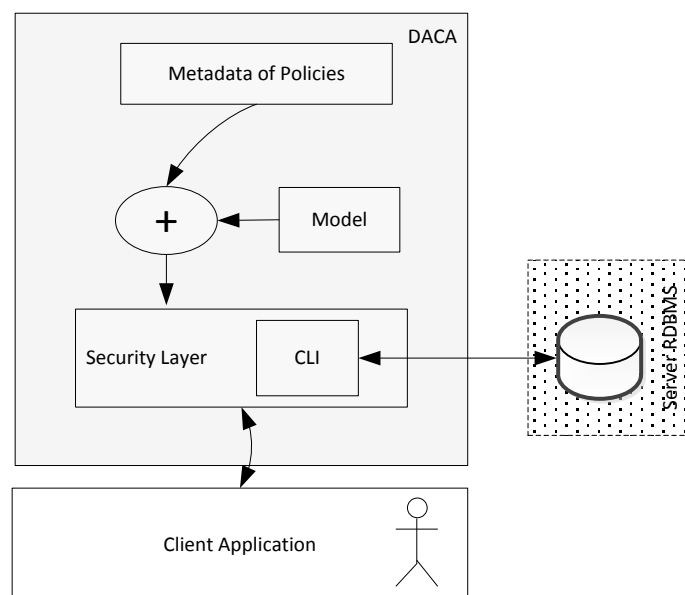


Figure 2. Simplified block diagram of the DACA.



There is no reference to FGACP. The DACA is focused on FGACM only. The DACA uses metadata derived from FGACP relying on any security model. This issue will be thoroughly addressed in chapter 4.

In order to validate the DACA, a solution based on Java, Java Database Connectivity (JDBC) and SQL Server was devised, designed and implemented. Two evaluations were carried out. The first one evaluates the DACA capability to address the announced drawbacks of current solutions, including the capacity to implement evolving FGACM dynamically, at runtime. The second evaluation is aimed at assessing its performance against a standard use of JDBC where no policies are enforced. The collected results show that the DACA is an effective solution to implement evolving FGACM on business tiers, of relational database applications, based on CLI, in this case JDBC.

### 1.3 Research questions

This thesis aims at answering several research questions related to the implementation of dynamic FGACM on business tiers of relational database applications. The research questions are derived from the issues described and emphasized in the previous sections.

The main question to be answered by this thesis is: is it possible to implement FGACM on business tiers dynamically, at runtime, and keep them updated when the policies evolve over time? If yes, the second level questions are:

#### Security

Current approaches allow users to write their own CRUD expressions freely. In an unsupervised context this opens possibilities to security violations. Is there any possibility to supervise the use of CRUD expressions effectively when protected data is being accessed?

#### Mastering of FGACP

Current security layers do not give any guidance on the established FGACP neither on the implemented FGACM. Is it possible to overcome this difficulty by providing programmers with a complete awareness about the established FGACM?

#### Use of CLI

The use of CLI to build business tiers presents several advantages. Is it possible to keep those advantages on the proposed solution to implement FGACM?

To answer these research questions, several steps need to be taken. In a first step, an architectural model is necessary aimed at addressing the main question of this thesis: the dynamic implementation of FGACM at runtime. In a second step, the use of CLI needs to be analyzed in order to implement FGACM. In a third step, a solution is necessary to convey a complete awareness to programmers about the implemented FGACM. Finally, in a fourth step, the proposed solution is evaluated to check its compliance with the research questions.

## 1.4 Contributions

The Dynamic Access Control Architecture is the main contribution of this thesis, exploiting CLI as the main standard API to be used. CLI are fundamental for the development process of business tiers whenever both a fine-tune control on the interactions with RDBMS and performance are considered key requirements. However, CLI do not provide any support for several software engineering challenges, such as how to implement dynamic FGACM on business tiers.

CLI are low level API and, as such, they convey some additional drawbacks when used for the building process of business tiers. Two of the most relevant drawbacks are:

### CLI are agnostic regarding schemas of database objects

CLI do not incorporate or provide any guidance about schemas of database objects. Programmers need to completely master schemas of database objects to be able to use CLI. This drawback deeply affects productivity of programmers during the development process and during the maintenance process of database applications. This drawback also hampers the enforcement of access control policies.

### CLI do not promote the reuse of software

Inefficiency of CLI to build reusable software is complete. Every business need impels programmers to write similar source-code to manage each CRUD expression. This drawback also hampers the development of reusable software for security layers.

Thus, these drawbacks have also a negative impact if security layers based on CLI are needed to enforce FGACP. FGACM control the access to database objects (formalized by schemas) and the DACA seeks to provide FGACM continuously updated and aligned with evolving FGACP. To address both drawbacks of CLI, some research was carried out. In a first step, the research was focused on defining a model to integrate schemas of relational databases with object-oriented applications using CLI. In a second step, the research was focused on defining an architecture for reusing adaptable business tier components relying on CLI. The dynamic implementation of FGACM was only addressed after the conclusion of these two lines of work. As such, in spite of not being considered as main contributions, the modelization process of business tiers and the componentization process of business tiers based on CLI are two cornerstones of the main contribution of this thesis.

### Modelization of business tiers

This research was focused on defining a model to integrate schemas of relational database objects with object-oriented applications when CLI are used to build business tiers [Pereira, '10b; Pereira, '11b]. The model defines typed objects aimed at managing the execution of CRUD expressions. A tool was also presented to ease the development process of typed objects from native CRUD expressions. Source-code of typed objects is automatically generated to be used on business tiers. In [Gomes, '11; Pereira, '10a] some research has also been carried out to evaluate the possibility of devising a high-performance version of the model, based on a thread-safe implementation.

Modelization of business tiers was the continuation of an earlier research focused on easing the development process of business tiers based on CLI but using stored procedures instead of

CRUD expressions [Óscar Mortágua Pereira, '05; Pereira, '06; Pereira, '07a; Óscar Narciso Mortágua Pereira, '05a; Óscar Narciso Mortágua Pereira, '05b].

#### Componentization of business tiers

Componentization addressed a key issue of defining an architecture for developing reusable and adaptable business tier components based on CLI. Several techniques were devised to address reusability and adaptation of business tier components. Among them, a technique was devised to deploy CRUD expressions at runtime which is one of the techniques used by the DACA to address evolving access control policies. The combination between the several techniques, next presented, led to the possibility of adopting several approaches for the building process of business tier components.

In [Pereira, '12d; Pereira, '11c; Pereira, '13f] a proposal based on a wide typed object is presented to support one specific business area at a time. Basically, a component is statically customized to support a business area, such as accountability, relying on a unique wide typed object. Then CRUD expressions are deployed at runtime in accordance with users' needs (eventually by access control policies). The typed object is said to be wide because it supports a schema for:

- All foreseen attributes to be returned from the database;
- All runtime values for column lists of all Update and Insert expressions;
- All runtime values for clause conditions of all CRUD expressions.

In [Pereira, '11a; Pereira, '13a] a component is also statically customized to support a business area but now relying on several typed objects. Each typed object addresses a specific business need such as implementing the reading process of attributes of a database object. CRUD expressions are also deployed at runtime in accordance with users' needs (eventually by access control policies). Each typed object supports all CRUD expressions whose schemas are in accordance with its own schema.

In [Pereira, '12b] a new customization process of business tiers is proposed. Here, customization is dynamically implemented at runtime, unlike the two previous approaches. Typed objects are dynamically created at runtime, following any of the two previous approaches. CRUD expressions are also deployed at runtime.

In [Pereira, '13b] an integrated perspective is given for multi-propose components based on CLI.

#### Access control

The security perspective is centered on access control and it is closely linked to the previous researches. In reality, the process to support evolving access control policies also includes the reusability and adaptability perspectives of business tiers components, which, by their side, include models.

[Pereira, '12d] presents an approach to address static implementation of FGACM based on CLI. In [Pereira, '12c] the previous approach was improved to address runtime adaptation of business tiers to implement FGACM. In [Pereira, '12a; Pereira, '13d] a complete and final perspective is given for an architecture to implement and keep FGACM updated on business tiers based on CLI. Additionally, to evaluate the impact of the enforcement mechanisms at the client side, a performance assessment was carried out. A scenario was defined and implemented to validate the approach. It is available through the Windows *Remote Desktop*

Connection” at: url: *ned.av.it.pt*, username: *DACA*; password: *guest* (only one user at a time is allowed to login).

## 1.5 Computational Tools and Infrastructure

Several computational tools and infrastructures were used since the first research on using stored procedures and CLI. Publications involving stored procedures, between 2005 and 2007, were based on the .NET framework and the following tools/technologies were used: Visual Studio 2005 (C#, ADO.NET, ASP.NET, Web Services), IIS (Internet Information Server) and SQL Server 2005. Publications since 2010 were based on the NetBeans (Java SE, Java EE, JPA, JDBC), Visual Studio 2010 (C#, ADO.NET, LINQ) and SQL Server 2008. Microsoft Northwind database was used in several works including the proof of concept of the DACA.

## 1.6 Thesis Organization

The remainder of this thesis is organized as follows.

Chapter 2 is divided in four main sections. The first, second and third sections provide the necessary background for a complete understanding of the technical aspects herein addressed. The fourth section is focused on the state of the art and presents some of the current approaches, commercial and academic, used to support access control.

Chapter 3 describes the evolution from CLI till the final DACA. It presents the modelization and the componentization approaches for CLI as key steps towards the DACA. Access control is also superficially addressed and a very concise presentation is made for the approach that has been followed.

Chapter 4 is dedicated to the DACA. It presents the methodology followed in this thesis to devise and design the DACA. It thoroughly presents and describes how the research was conducted. Beyond the information to convey the believability of the obtained results, this chapter provides the necessary information to allow other researchers replicate and design accurately solutions based on the DACA.

In Chapter 5, a scenario based on the DACA is defined and implemented to evaluate the DACA against the announced research questions. This chapter is divided in three sections. The first section is aimed at implementing a scenario based on the DACA. The second section is aimed at assessing the DACA performance against a standard use of CLI but without any FGACM. The third section is aimed at analyzing the collected results to evaluate if the DACA answers the research questions of this thesis positively.

Chapter 6, the final chapter, is organized in four sections. The first section is focused on presenting an overview of this work. The second section is focused on presenting the contributions of this work. The third section concisely discusses some important issues closely related to this thesis but out of its scope. Finally, the fourth section presents the future work to be conducted to continue the work here presented.

## 2 Background and State of the Art

This chapter is focused on presenting the necessary background and the state of the art in the area of access control to promote a good understanding on this thesis contents and to make it a self-contained document for most of the readers. It comprises five sections each one addressing a particular subject: section 1 presents the basic access control concepts; section 2 presents current tools for building relational business tiers; section 3 presents JDBC; section 4 presents current approaches to access control, general techniques and related work and, final, section 5 summarizes to contents of this chapter.

This chapter is organized as follows. Section 2.1 presents the basic access control concepts. Section 2.2 presents the current tools that are used for building business tiers. Section 2.3 introduces the JDBC which is the a key API of the DACA. Section 2.4 presents the state of the art and, finally, section 2.5 summarizes the content of this chapter.

### 2.1 Basic Access Control Concepts

This section provides the required background to completely understand the fundamental concepts and techniques of access control and also the one used on the DACA.

Access control is a concept used in several applications and several contexts. It is focused on preventing unauthorized accesses to protected resources. Access control is enforced by security layers, which mediates every attempt to access to protected resources. Access control has been used on several situations, such as to control the access to: XML documents [Bertino, '00; Damiani, '02; Fundulaki, '04; Iwaihara, '05; Luo, '04], Web Services [Bhatti, '05; Koshutanski, '03; Mecella, '06; Paci, '11; Sharifi, '09; Wonohoesodo, '04], publish/subscribe systems [Belokosztolszki, '03], social networks [Anwar, '12; Carminati, '09a; Carminati, '06; Carminati, '09b], pervasive computing systems [José, '09; Kim, '09; Kulkarni, '08; Vagts, '11; Zhang, '03], content shared in the web [Tootoonchian, '08], collaborative environments [He, '09; Hildmann, '99; Raje, '12; Tolone, '05], grid computing system [Oo, '07; Wang, '06; Zhang, '03], cloud computing systems [Zhu, '12], sensor networks [Garcia-Morchon, '10; Hur, '11; Liu, '10; Vuran, '06; Ye, '04] and mobile communications [Lawson, '12]. In this thesis we are focused on protecting data residing inside and managed by RDBMS. In this context, access control is aimed at limiting the activities of legitimate users (legitimate at the database level) to access sensitive data residing in RDBMS. Authentication and access control concepts must not be confused. The authentication process is responsible for identifying database users while access control assumes that a previous authentication of users has been accomplished before enforcing any security policy. Access control has a specialized branch dedicated to privacy protection [Shi, '09; Wang, '07], which is generally known as fine-grained access control (FGAC). Unlike general access control, which is concerned on providing protection to

data at the table and view level, FGAC is concerned with providing ways to control the access to protected data at the row and even at the cell level.

### 2.1.1 Access Control Strategies

Security policies define rules through which access control is governed. Three of the main strategies for regulating access control policies are [Samarati, '01a; Vimercati, '08]: discretionary access control (DAC) [Sandhu, '94], mandatory access control (MAC) and Role-based access control (RBAC) [Ferraiolo, '01; Sandhu, '00]. There are other strategies for regulating access control, such as attribute-based access control (ABAC) [Kuhn, '10], credential-based access control (CBAC) [Li, '05; Yu, '03], content driven [Moffett, '91; Staddon, '08], location driven [Decker, '08], public key driven [Wang, '11] and certificate driven [Samarati, '01b]. Each one addresses specific security needs for the system under protection. Next follows a description for the three main policies: DAC, MAC and RBAC.

#### 2.1.1.1 Discretionary Access Control Policies

DAC [Vimercati, '08] is based on the identity of users and on the access rules stating what users are and are not allowed to do when they request access to a protected resource. DAC is based on a set of rules, known as authorizations, which state which user can perform which action on which resource. In the most basic form, an authorization is a triple  $(s, a, r)$ , stating that subject (user)  $s$  can execute action  $a$  on a resource  $r$ . The first discretionary access control model proposed in the literature is the access matrix model [Graham, '72; Harrison, '76; Lampson, '74]. Table 1 shows an adaptation of the standard access matrix concept assigning an access matrix to two users (A and B) to execute actions (Read, Update, Insert and Delete) on a database table with attributes  $a, b, c$  and  $d$ . This access matrix defines, for each table attribute (resource) and for each user, which actions (read, update and insert) each user is authorized and is denied to perform. *Delete* action is authorized in a tuple basis and, therefore, it is executed as an atomic action on all attributes as shown in Table 1.

User	Action	a	b	c	d
A	Read	yes	no	yes	yes
	Update	no	yes	no	yes
	Insert	yes	yes	no	no
	Delete	yes			
B	Read	no	no	yes	yes
	Update	yes	yes	no	yes
	Insert	yes	yes	no	no
	Delete	no			

Table 1. Access matrix to a table with attributes  $a, b, c$  and  $d$ .

DAC presents several security vulnerabilities. DAC does not separate the concept of *User* from the concept of *Subject*. When using DAC policies, Users are actors authorized to run a system and to whom permissions are granted. To run any system, processes (subjects) are created on behalf of

users. As DAC policies do not consider the distinction between Users and Subjects, DAC policies evaluate permissions of subjects according to permissions of their users. This security gap is used by malicious programs to exploit the legitimate permissions of users, as it happens with Trojan Horses [Samarati, '01b].

### 2.1.1.2 Mandatory Access Control Policies

MAC [Samarati, '01b; Vimercati, '08] enforces access control on the basis of regulations mandated by a central authority. The access to protected resources is governed on the basis of classification of subjects and resources on the system where each one has an assigned security level. The security level assigned to a resource measures its sensitivity. The security level assigned to a user, called clearance, measures its reliability to access protected resources. The most common form of mandatory policy is the *multilevel security policy*. Unlike DAC, MAC policies distinguish users from subjects and the access control is enforced on processes operating on behalf of users. Each subject and resource is associated with an *access class*, usually composed of a *security level* and a set of *categories*. Security levels in the system are characterized by a total order relation, while categories form an unordered set. As a consequence, the set of access classes is characterized by a partial order relation, denoted  $\geq$  and called dominance. Given two access classes  $c_1$  and  $c_2$ , class  $c_1$  dominates class  $c_2$ , denoted  $c_1 \geq c_2$ , if and only if the security level of class  $c_1$  is greater than or equal to the security level of class  $c_2$  and the set of categories of class  $c_1$  includes the set of categories of class  $c_2$ . Access classes together with their partial order dominance relationship form a lattice [Sandhu, '93].

### 2.1.1.3 Role-Based Access Control Policies

RBAC is the most popular access control policy to protect data residing in relational databases. As such, RBAC is described in more detail than DAC and MAC.

RBAC [Ferraiolo, '92; Sandhu, '96] decisions are based on the roles that individual users play on an organization such as hospital administrator, doctor and nurse. RBAC is attracting increasing interest particularly of vendors of database management systems, and a standardization was proposed by NIST (National Institute of Standards and Technology) [Sandhu, '00]. A role is defined as a set of permissions associated with the subjects (users) playing that role. When accessing the system, each subject has to specify the role he/she wishes to play and, if he/she is granted to play that role, he/she can exploit the corresponding permissions. A permission is an authorization to execute an operation in a protected resource. Thus, permissions are assigned to roles and roles are assigned to subjects. The access control policy is then defined through two different steps: firstly, the administrator defines roles and the permissions related to each of them; secondly, each subject is assigned with the set of roles he/she can play. Roles can be hierarchically organized to exploit the propagation of access control privileges along the hierarchy. This approach is a natural means for organizing roles to reflect lines of responsibilities in organizations. Each user may be allowed to play more than one role simultaneously and more users may play the same role simultaneously, even if restrictions on their number may be imposed by the security administrator. It is important to note that roles and groups of users are two different concepts. A group is a named collection of users and possibly other groups. A role is both a named collection of users on one side and collection of permissions on the other side. Roles serve as linking entities to bring permissions and

users together. Furthermore, while roles can be activated and deactivated directly by users at their discretion, the membership in a group cannot be deactivated. The main advantage of RBAC, regarding to DAC and MAC, is that it better suits commercial environments. In fact, in a company, the identity of a person is not important for his/her access to the system, but his/her responsibilities are. Also, the role-based policy tries to organize privileges mapping the organization's structure on the roles hierarchy used for access control.

RBAC is commonly ruled by three security principles: least privilege, separation of duties and data abstraction. Next follows a description for each security principle.

#### Least Privilege

The least privilege principle requires that a subject be given no more privilege than necessary.

Least privilege is used to ensure that only those permissions required to accomplish a task carried out by subjects of a role are effectively assigned to that role.

#### Separation of Duties

The separation of duties principle requires that mutually exclusive roles must not be granted to the same subjects to complete sensitive tasks. For example, the authorization to complete a task should not be given by who is requesting the authorization to complete the requested task. Separation of duties is accomplished statically or dynamically. Static separation of duties is enforced through constraints on the assignment of users to roles. Dynamic separation of duties is enforced by placing constraints on the roles that can be activated within a user's session.

#### Data Abstraction

The data abstraction principle is supported by means of abstract permissions such as *credit* and *debit* for an account rather than the usual low level permissions such as read and write permissions.

In spite of its importance, RBAC does not solve all access control issues. In situations where access control is required to deal with sequences of operations, additional access control mechanisms are often required. For example a purchase requisition may require several intermediate steps before being a purchase order.

Some of the vendors offering RBAC security environments on their products are: Microsoft, Cisco Systems, IBM, Siemens, Symantec, Sybase and Oracle.

## **2.1.2 Architectures for Access Control Mechanisms**

Access control is usually implemented in a three phase approach [Samarati, '01b]: security policy definition, security model to be followed and security enforcement mechanisms. This thesis is focused on access control mechanisms and, therefore, an overview of the architectures for their implementation is presented and described in this sub-section.

Access control mechanisms implement the security policy formalized by the security model. In this thesis we are focused on providing access control to data residing in RDBMS and specially FGAC. Several architectural approaches are available to implement FGACM to protect data in RDBMS. Some are provided by the vendors of RDBMS and others have been proposed by the



research community. Very often, access control mechanisms comprise a runtime procedure, known as decision evaluation, to evaluate if permissions are granted or denied. Basically there are three main architectural approaches:

#### Centralized approach

In the centralized approach decisions about granting or denying access, and access control mechanisms are both managed by centralized entities. This is the approach used by vendors of RDBMS.

#### Distributed approach

In the distributed approach decisions about granting or denying access, and access control mechanisms are both locally managed on the client-side applications. The DACA is based on this approach.

#### Mixed approach

In the mixed approach decisions about granting or denying access are managed by a centralized entity but the access control mechanisms are managed on the client-side applications. The most well-known example is the eXtensible Access Control Markup Language (XACML) [OASIS, '12] where policies are enforced in the client-side by Policy Enforcement Points (PEP) of database applications but the decision whether to grant or deny access is taken by centralized Policy Decision Points (PDP).

The following sub-sections present a more detailed description for each architectural approach just described.

### 2.1.2.1 Centralized Approach

The centralized approach is based on a security layer developed by security experts and usually using RDBMS tools and based on RBAC policies. Access control policies 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 for software applications. Their presence is only noticed if some unauthorized access is detected by the security layer. Figure 3 presents a simplified block diagram for the centralized approach. Basically, SQL statements are sent to the RDBMS (Figure 3: 1) and before being executed they are evaluated by a security layer to check their compliance with the established access control policies. If any violation is detected, SQL statements are rejected and an exception is raised, otherwise they are executed (Figure 3: 2).

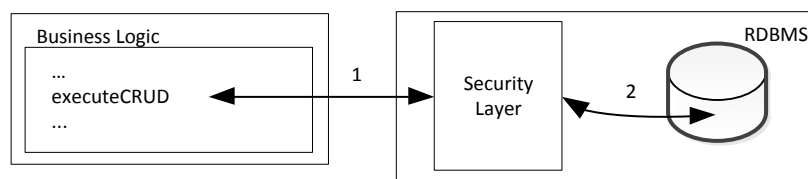


Figure 3. Centralized access control mechanism.

The centralized approach presents advantages and disadvantages. Among the advantages, the following are emphasized:

#### Maintenance

Whenever a maintenance activity is necessary, it is restricted to the entity responsible for the decisions and the mechanisms. The centralized approach clearly avoids the need to carry out maintenance activities in several equipment.

#### Reliable security

If security is correctly defined and implemented, any attempt to violate the system security is always evaluated against the enforced access control policies.

Among the disadvantages [Lopez, '02b; Valle, '02], the following are emphasized:

#### Scalability

Access control always conveys an additional processing overhead and, in case of complex decisions and mechanisms, there may be scalability problems.

#### Increased latency

Requests to access to data, and the decisions and mechanisms to control those accesses reside in different computer devices leading to an increased latency when the access is not granted. On the distributed architectures, decisions and mechanisms are local and, therefore, the latency is minimized.

#### Single point of failure

As any centralized architecture the single point of failure may lead to undesirable security failures. In this unwanted security failure, every request to access protected data may exploit the security gap.

### **2.1.2.2 Distributed Architecture**

A distributed approach can be characterized by the distributed character of the decisions making and also on the distributed character of the enforcement mechanisms. Decisions and mechanisms are implemented and placed in each running client-side application. Not only the mechanisms are distributed but also the decisions are locally taken. Such distributed architecture approach conveys some advantages and some disadvantages. Among them, the following advantages are to be emphasized:

#### Scalability

The rational to decide upon granting or denying access and the mechanisms are deployed in each client-side application. This approach clearly delegates in each client the total responsibility to ensure and to comply with the established access control policies. When the number of client-side applications increase, there is no effect on the performance and on the responsiveness of the extended system. If complexity of the decisions and/or mechanisms increases, the additional power computation that is needed is not cumulative in any centralized equipment but distributed in each client-side equipment. Therefore, each client-side equipment

has the responsibility to provide the eventual necessary additional computational power to avoid any security violation. This issue has an increased relevancy because very often database servers are bottlenecks in intensive database applications. If beyond the access to data they are also required to provide access control, then very probably the bottleneck will be more noticeable. The distributed architecture clearly relieves database servers from the responsibility of providing access control.

#### Minimum latency

In the distributed architecture decisions and mechanism are deployed in each client application and, therefore, the latency for any request to access protected data is minimized. In non-distributed architectures, latency may be significant when requests to access to data are denied. While in distributed architectures the decision is made locally, in non-distributed architectures there is the unavoidable latency for the communication process between client applications and the centralized security equipment.

Among the disadvantages the following are next emphasized:

#### Maintainability

Whenever a maintenance activity is carried out on the access control policies, it is potentially necessary to extend the maintenance activities to all equipment running client-side applications. If the maintenance activity is not automated then it may convey a huge effort in systems comprising many and faraway client equipment. However, this potential disadvantage is not applicable to the DACA as it has already been mentioned. The DACA has an automated process to keep FGACM updated in all client-side equipment.

#### Security gap

If policies are not coordinated from a central point, the probability of deploying security gaps is increased. This potential disadvantage is not applicable to the DACA because the DACA comprises central systems responsible for ensuring that the implementation of FGACM in all client-side equipment is in accordance with the established FGACP.

### **2.1.2.3 Mixed Architecture**

A mixed approach splits the responsibilities for the access control between a centralized server and client-side equipment. The best well-known standard is the XACML. XACML is an access control language based on XML and defined by the Organization for the Advancement of Structured Information Standards (OASIS). The basic design of an XACML system has four main components: PAP (Policy Administration Point), PEP (Policy Enforcement Point), PIP (Policy Information Point) and PDP (Policy Decision Point). We will be mainly focused on the PEP and PDP components but PAP and PIP will also be described.

The XACML approach consists in a security software layer with two main functionalities: the PDP and the PEP, as defined in XACML [OASIS, '12] and used in [Corcoran, '09], see Figure 4. The PEP are locally inserted in-line with the client-side source code to intercept users requests for accessing a resource protected by an access control policy (Figure 4: 1) and enforces the decision to be evaluated by a remote PDP on this access authorization. PDP evaluates requests to access a resource against the access control policies to decide whether to grant or to deny the access

(Figure 4: 2). If authorization is granted, PEP uses business logic to perform the authorized action (Figure 4: 3) and, if no other restriction exists, the action is executed by the RDBMS (Figure 4: 4). PEP are intentionally inserted in key points of the source code to enforce PDP decisions. PEP-PDP approach is a mixed approach involving the centralized approach and the distributed approach. Beyond PEP and PDP components, the basic design of an XACML system has two more main components: PAP and PIP. PAP is where administration of policies is carried out. PIP is where information is collected for the PDP to make up decisions. The advantages and disadvantages of the mixed architecture emanates from the advantages and disadvantages of the centralized and distributed architectures. Anyway, latency deserves a closer attention. In the centralized approach, when compared with the distributed approach, the latency is only noticeable if permission to access the protected resource is not granted. In the mixed approach, the latency is permanent and independent from the decision process. This means that every request to a protected resource entails a latency to evaluate if permission is or is not granted.

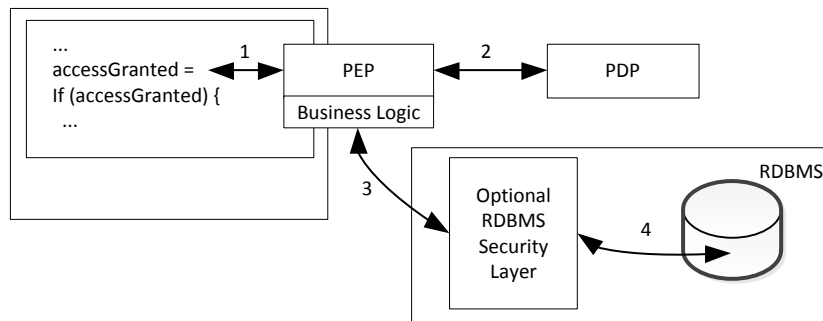


Figure 4. Mixed architecture based on PEP and PDP.

### 2.1.3 Dimensions of Access Control Mechanisms

In this sub-section a survey is made about the main dimensions that may influence the implementation of access control mechanisms.

The enforcement of access control policies comprises five orthogonal dimensions: architecture, granularity, awareability, contextuality and adaptability. The architectural dimension, because of its visibility and notoriety, has been already presented separately in sub-section 2.1.2. The remaining four dimensions are now jointly presented and described in this sub-section. Granularity is focused on characterizing the granularity of the data to be disclosed and also the authorized actions on it. Awareability is focused on evaluating if access control mechanisms are or are not made available to programmers while they write source code for client-side applications. Contextuality is focused on evaluating if access control uses runtime context to decide upon the data to be made available and the authorized actions. Adaptability is focused on evaluating if access control mechanisms are or are not automatically updated when access control policies evolve over time. Next follows a detailed description for each dimension.

### 2.1.3.1 Granularity

Currently, most of the RDBMS provide access control mechanisms driven by fixed relations between users, operations and tables leading to a maximum protection of resources at the column-level [Caires, '11]. This level of protection may be satisfactory in many situations but in many others it is far from being satisfactory. In situations where the access to data is not controlled at the column level but is also controlled at the row level, it is considered as being a FGACP. Some RDBMS vendors support FGACP using different approaches, such as query rewriting in INGRES, Virtual Private Database [Oracle] in Oracle and label-based in DB2 [Bond, '07]. Very often these features of RDBMS are not enough and there is the need to provide an increased level of protection. This need for an increased level of protection has been the motivation for the researches that have been conducted and also one of the key motivations of this thesis.

### 2.1.3.2 Awarability

Programmers of client applications can hardly master established access control policies in database applications with many and complex policies. As such, it is convenient to follow an approach where the policies are statically checked at development time or at compile time of applications tiers. This awarability relieves programmers from mastering FGACP and, additionally, conveys a swift feedback about any policy violation. This approach conveys two important advantages:

#### Productivity

The obligation to evaluate the correctness of source-code at runtime induces an additional overhead at the development process, this way leading to a decrease in productivity. Therefore, productivity is improved if programmers are relieved from running applications to become aware of any disconformity with the established access control policies;

#### Security

Due to the endless SQL expressiveness, difficulties arise to ensure a complete secure access control. To overcome this fragility, a possible approach is by implementing a fine control on the set of CRUD expressions that are allowed to be executed to improve the overall security. Ultimately, CRUD expressions are provided by database administrators and are statically made available to programmers, this way avoiding the free writing process of CRUD expressions by programmers of business tiers and application tiers.

The awareness of access control policies follows two approaches, the typed approach and the untyped approach:

#### Typed approach

The typed approach is based on typed objects to enforce access control policies so that programmers become aware of their existence at development time. The awareness at development time is the approach that improves productivity more. Programmers become aware of any disconformity while writing the source code, this way avoiding the waste of time for writing erroneous source code and the waste of time for compiling and re-writing the source code.

### Untyped approach

The untyped approach is based on data structures used by the compiler to check if the accesses to data are in conformity with the established access control policies. Programmers become aware of any unconformity only after writing and compiling the source code, which, obviously, leads to a lower productivity than the typed approach.

The typed approach has the enormous advantage of guiding programmers in the correct direction while editing source-code. This guidance is automatically provided in current Integrated Development Environments (IDE), such as NetBeans [Oracle, '12e], Eclipse [Eclipse, '12] and Visual Studio [Microsoft, '10], through the auto-completion facility. Awareness at compile-time does not guide programmers during the editing process and postpones the feedback about any unconformity until the compilation process of source-code is completed.

### **2.1.3.3 Contextuality**

There are situations where access control is governed by contextual information. In these situations, there is no possibility to know at development time or even at compile time the values to be used to protect the access to sensitive data. Typical situations are health care organizations and social networks. In health care organizations, patient data is disclosed only to the people who need them and have the correspondent authorization. For example, a doctor can access the data only of those patients he is treating. In social networks, only current friends have access to some data. In such cases, before being executed, CRUD expressions must comprise the required predicates to avoid the access to unauthorized data. The predicates may be originally written with CRUD expressions or later added using any query rewriting technique. The query rewriting technique is presented and described in sub-section 2.4.1. A concrete case of this approach is the Reflective Access Control [Olson, '08]. A “...a policy is defined as reflective when it depends on data contained in other parts of the database” [Olson, '09].

### **2.1.3.4 Adaptability**

Access control policies define which actions each user is authorized to execute on database objects. Nothing prevents the policies established during a certain period to evolve to a different state. When this happens, users are allowed to do things they were not allowed to do before, or users are not allowed to do anymore things they were allowed to do before or users are allowed to do what they were allowed to do before but in a different way. To guarantee that this process of evolving policies is supported, it is necessary to guarantee that the associated mechanisms and decisions are also updated. If we recall the architectural dimension we can see that very probably some difficulties arise. The centralized architecture requires maintenance activity in the central system, which does not raise any special concern in a first glance. But the distributed and mixed architectures entail a maintenance activity in all client equipment where the policies are enforced, which may raise justified concerns. Two approaches are followed to implement access control mechanisms, the static approach and the dynamic approach.

### Static approach

In the static approach, mechanisms and/or decisions are hard coded and there is no way to

automatically modify them in accordance with new policies. Therefore, whenever modifications are needed, they have to be edited, compiled and manually deployed. This updating process is not scalable leading to a huge effort when database applications have many client-side equipment.

### Dynamic approach

In the dynamic approach, mechanisms and/or decisions have the ability to be adapted to evolving policies. While mechanisms are traditionally hard coded, the decisions may also resort to a database to become adapted more easily. The adaptation process of hard coded components of mechanisms and decisions may rely on different strategies but always entails two automated procedures: building/adaptation of source-code and a deployment process. The combination of these two procedures opens the possibility to opt for two different implementations: 1) policies are deployed and adapted in each server (decisions) and each client-side system (decisions and/or mechanisms) or 2) mechanisms and decisions are adapted and then deployed into each security sever (decisions) and each client-side system (decisions and/or mechanisms). Unlike the static approach, the dynamic approach follows an automated process for adapting policies, this way promoting scalability, maintainability and productivity.

## **2.2 Current tools for Building Business Tiers**

Several tools have been devised to improve the development process of business tiers mainly for tackling the impedance mismatch issue [David, '90]. From them, two categories have had a wide acceptance in the academic and commercial forums: Object-to-Relational Mapping tools (O/RM) and CLI. Other solutions, such as embedded SQL [Moore, '91] (SQLJ [Eisenberg, '99]), have achieved some acceptance in the past but failed to be generally accepted by the research and commercial communities. Others were proposed but without any general known acceptance: Safe Query Objects [William, '05] and SQL DOM [Russell, '05]. These tools were all devised mainly to tackle the impedance mismatch issue not addressing the concept of access control at all. Access control, whenever implemented, is based on additional security layers. Some examples based on current tools are shown to demonstrate their inability to deal with access control. Firstly, O/RM tools are presented because of their importance and their wide acceptance. Then, CLI will be presented and described because of their undeniable relevance in the DACA.

### **2.2.1 O/RM tools and ADO.NET**

O/RM tools [Keller, '97; Lammel, '06], such as LINQ [Erik, '06], Hibernate [Christian, '04], Java Persistent API (JPA) [Yang, '10], Oracle TopLink [Oracle], CRUD on Rails [Vohra, '07], and ADO.NET were designed to create, in the object-oriented paradigm, static representation models of relational database schemas. The static model is built in a first stage, eventually by a database administrator, and then programmers start the development process. The basic artifacts of the static representation models are classes (entities), each one representing a database table. Through these entities programmers may read data from tables, update data, insert new data and, finally, delete existing data. To support explicit CRUD expressions, O/RM tools provide proprietary SQL languages. Despite these advantages, O/RM do not address access control at any level. Additionally, O/RM tools lead to some drawbacks, such as: 1) they induce an additional overhead when

```

26 private void useADO()
27 {
28     String sql = "Select * from Products where productId=10";
29     SqlDataAdapter da = new SqlDataAdapter();
30     da.SelectCommand = new SqlCommand(sql, conn);
31     SqlCommandBuilder cb = new SqlCommandBuilder(da);
32     DataSet ds = new DataSet();
33     da.Fill(ds, "Products");
34     DataRow dr = ds.Tables["Products"].Rows[0];
35     productName = (String) dr["productName"];
36     // ... more code
37     dr["productName"] = productName;
38     cb.GetUpdateCommand();
39     da.Update(ds, "Products");
40     // ... more code
41 }

```

Figure 5. Example based on ADO.NET.

```

33 private void useJPA() {
34     TypedQuery<Products> q= em.createNamedQuery( "Products.id_10", Products.class );
35     List<Products> prd = q.getResultList();
36     for ( Products p: prd ) {
37         productName=p.getProductname();
38         // ... more code
39         p.setProductname(productName);
40         em.persist(p);
41         // ... more code
42     }

```

Figure 6. Example based on JPA.

```

15 private void useLINQ()
16 {
17     Product prd = dc.Products.Single(p => p.CategoryID == 10);
18     productName = prd.ProductName;
19     // ... more code
20     prd.ProductName = productName;
21     dc.SubmitChanges();
22     // ... more code
23 }
24

```

Figure 7. Example based on LINQ.

compared to CLI; 2) they were not devised having in mind the frequent use of complex CRUD expressions and, finally, 3) they rely on static models, this way not promoting an easy process for a dynamic adaptation at runtime. Moreover, O/RM tools do not promote a clear separation of application tier developer role from business tier developer role. For example, programmers may use embedded language extensions and other embedded functionalities to extend pre-built static models, this way opening possible security gaps. Figure 5, Figure 6 and Figure 7 present a simplified version of the example presented in Figure 1 but written in ADO.NET, JPA and LINQ, respectively. Akin to JDBC, programmers are free to write any CRUD expression (Figure 5: line 28, Figure 6: line 34, Figure 7: line 17) and to execute them. Then, they have no restrictions to read the attribute *productName* (Figure 5: line 35, Figure 6: line 37, Figure 7: line 20)



not even to update it (Figure 5: line 37-39, Figure 6: line 39-40, Figure 7: line 20-21). Beyond reading and updating the attribute *productName*, it is also possible to read and update all the remaining attributes, even to insert new rows and delete existent rows. There is no possibility to prevent programmers from issuing these accesses. This example clearly shows the unpreparedness of current O/RM tools to deal with access control policies at any security level.

## 2.2.2 Call Level Interfaces

CLI are the main API to model one of the most important components of the DACA. The component is responsible for the implementation of the dynamic FGACM at the client-side of database applications. As such, a detailed knowledge about the architecture and features of CLI is considered essential to understand the DACA and also the options made for the implementation of FGACM.

### 2.2.2.1 Overview of Call Level Interfaces

CLI are an ISO/IEC standard [ISO, '03] for the interaction between RDBMS and client applications. Two API were initially devised for the C and COBOL programming languages. The most well-known implementation of CLI is the Open Database Connectivity (ODBC) [Microsoft, '92] specification. ODBC is a C programming language interface providing a standard for client applications access data from a variety of RDBMS. “*ODBC is a low-level, high-performance interface that is designed specifically for relational data stores.*” [Microsoft, '92]. In this document, the term CLI is used with a wider scope than the one defined by ISO/IEC. Herein, CLI are used to refer to any API/standard with identical features and characteristics to the standard emanated from ISO/IEC. In this context, other related CLI have also been devised, such as JDBC. Other tools/frameworks have also been devised to ease the development process of business tiers, which, in most of the situations, use CLI as the underlying technology to interact with RDBMS, such as ADO.NET [Mead, '11], JPA [Yang, '10] and Hibernate [Bauer, '07]. Some of the main features of CLI that are important for the DACA are now briefly described:

#### Building process of CRUD expressions

CRUD expressions are the main entities used by programmers to interact with data residing in RDBMS. Thus, the key issue of any tool devised to develop business tiers is the definition of how client applications build and use CRUD expressions. CLI allow CRUD expressions to be written in the native SQL language and encoded inside strings. There is no layer between the native SQL language and the services provided by CLI.

#### Access Modes

CLI provide several access modes to data residing in RDBMS. Programmers are free to select at any moment the access modes that more effectively address their needs.

#### Results of CRUD expressions

From the application's perspective, every CRUD expression has a final result. Insert, Update and Delete expressions modify the state of databases by affecting a certain number of rows. Select expressions create a set of rows that must be made available to client applications.

### Performance optimization

CLI provide several performance contexts in which CRUD expressions may be executed. Client applications are called to choose in each situation the context most appropriated.

## **2.2.2.2 Functionalities**

CLI are considered important options for building business tiers whenever a fine tune control on the interactions with RDBMS is necessary and also when performance is considered a key requirement [Cook, '05]. This is confirmed in several functionalities of CLI, being the diversity of access modes just one example. CLI provide several access modes to data residing on RDBMS among them the possibility to encode CRUD expressions inside strings, this way easily incorporating the power and the full expressiveness of the SQL language. JDBC [Parsian, '05], for Java environments, and ODBC [Microsoft, '92], for Windows environments, are two representatives of CLI. CRUD expressions are executed against the host database and the possible results they produce (only for Select expressions) are locally managed by local memory structures (LMS) – (ResultSet [Oracle, '13] for JDBC, RecordSet [Microsoft, '13] for ODBC). CLI provide two main and key functionalities to access data:

### Use of the native SQL language

This functionality has been already described. CLI are suited to the use of the native SQL language. This way, they are prepared to exploit the performance and the full expressiveness of the SQL language.

### Use of LMS

LMS are containers prepared to help client application to interact with data returned by Select expressions. They provide services to allow applications to read, insert, update and delete data from the LMS.

Only services of CLI directly related to the execution of CRUD expressions will be addressed in this thesis. Services such as those for managing connections to host databases are not here addressed. Main services of CLI are organized in four main categories: execution, scrollability, updatability and transactions.

### Execution

Execution comprises services related to the execution of native CRUD expressions. Native CRUD expressions are executed as compiled-on-the-fly or pre-compiled (when they are to be reused). Pre-compiled CRUD expressions are stored in the database by RDBMS while being used and, therefore, very often provide a much better performance execution than those compiled-on-the-fly. Additionally, CLI deal differently with Select expressions from the other three types of CRUD expressions. Select expressions instantiate LMS, while the other types do not. These latter types (Insert, Update and Delete) generate a value indicating the number of affected rows in the database.

### Scrollability

Scrollability comprises services related to the scrolling process on LMS. There are several

different implementations but two are emphasized. They are mutual-exclusive and are herein known as forward-only and scrollable.

#### Forward-only

Forward-only LMS restrict the possibility to move cursors one row forward at a time. Forward-only LMS are used when rows are accessed in a sequential way, one by one, from the first one till the last one.

#### Scrollable

Scrollable LMS do not restrict the movement of cursors. Unlike the forward-only LMS, programmers are free to select rows not placed next and after the active selected row. Programmers are free to write source code to jump several rows at a time and in any direction, forward or backward.

The choice between forward-only and scrollable LMS not only affects the functionalities of LMS but also their performance. This issue will be addressed during the performance assessment.

#### Updatability

Updatability comprises services organized in protocols to interact with data contained in LMS. There are several implementations but two are herein emphasized. They are mutual-exclusive and are known as read-only and updatable LMS.

#### Read-only LMS

Read-only LMS restrict the access to their in-memory data to read operations only. Applications are prevented from inserting new rows, from updating existent rows and also from deleting existent rows on LMS.

#### Updatable LMS

Updatable LMS do not restrict any operation on their in-memory data. Applications are allowed to read existent rows, insert new rows, update existent rows and to delete existent rows. The important aspect in these actions is that CLI, internally, create CRUD expressions to execute the actions performed at the LMS level. Thus, when an update protocol is committed, CLI create an Update expression to update the updated attributes. Similarly, when a row is inserted or deleted at the LMS level, CLI create Insert and Delete expressions to materialize the requested actions, respectively. These additional actions at the LMS level avoid the need to write native CRUD expressions to perform the equivalent actions on host databases.

The choice between read-only and updatable LMS not only affects the functionalities of LMS but also their performance as it will be shown.

#### Transactions

Transactions comprise a set of services to manage database transactions such as save points, rollback transactions and commit transactions. CLI provide two mutual-exclusive modes to manage database transactions: auto-commit mode and normal mode.

Auto-commit mode

The auto-commit mode is used when there is the need to execute each CRUD expression atomically and, as such, ruled by an individual transaction. This mode avoids the need to precede and follow each CRUD expression by a transaction initiation process and by a transaction commit process.

Normal mode

The normal mode is used when there is the need to process several CRUD expressions as a single transaction. In this case, the first CRUD expression is preceded by the beginning of a transaction and the last CRUD expression is followed by the correspondent transaction commit.

By default, the work mode is the auto-commit mode on which each CRUD expression is automatically committed when it is completed successfully. Therefore, in this mode, no other transaction management functions are required. In the normal mode there are functions to deal with traditional transaction actions, such as begin transactions, commit transactions and rollback transactions.

**2.2.2.3 Local Memory Structures**

LMS have been loosely presented and some properties have also been already described. Next follows a more detailed description about the operation of LMS.

LMS are instantiated to manage the data returned by Select expressions. As such, at this point it is advisable to discuss some LMS features that are relevant to this research. Figure 8 presents a general LMS containing 5 tuples (rows, 1 to 5) and 6 attributes (a, b, c, d, e, f). This LMS could have been instantiated to manage the data returned by the following CRUD expression: *Select a, b, c, d, e, f from Table Where ....* In this case, the CRUD expression has returned 5 tuples and the current selected tuple is tuple number 2. Two representatives of LMS are ResultSet [Oracle, '13] (JDBC) and RecordSet [Microsoft, '13] (ODBC).

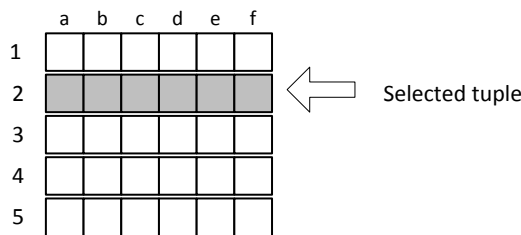


Figure 8. LMS with 5 tuples (rows) and 6 attributes (a till f).

The access to LMS attributes is accomplished by selecting a tuple and then, through an index or through a label (usually the attribute name), by selecting one attribute at a time. For example, to execute an action *action* (read, insert or update) on attribute *c* of tuple 2:

- Select tuple 2;
- Execute *action(index of attribute c)* or *action(label of attribute c)*.

CLI are responsible for providing services to allow applications to scroll on LMS, to read their contents and to modify (insert, update, delete) their internal contents (other services are also available but they are not relevant at this point). Services may be split in two categories: basic services and advanced services. Basic services comprise two groups of protocols: the scrolling protocols are aimed at scrolling on tuples and the read protocol is aimed at reading the tuples' attributes. Advanced services are available only if LMS are updatable. In this case applications are allowed to change the internal state of LMS. Advanced services comprise three protocols: insert protocol to add new tuples, update protocol to update an existent in-memory tuples and, finally, delete protocol to delete existent tuples. After being committed, new LMS states are automatically committed in the host database. To execute any of the previous services it is necessary to know that the access to LMS is simultaneously tuple oriented and protocol oriented. This has two main implications. First, at any time only one tuple may be selected as the target tuple. Second, if a protocol is being executed, applications should not start any other protocol. If this rule is not fulfilled, LMS may lose their previous states. For example, if an advanced service is being executed and another protocol is triggered, LMS discard all changes made during the first protocol. Table 2 concisely presents four of the five main LMS protocols. Scroll is not presented because only the presented protocols are used to interact with data managed by LMS and, therefore, managed by RDBMS. Additionally, the scroll protocol is orthogonal to the remaining four protocols.

#### Read Protocol

During the read action, attributes are individually read one by one and always from the current selected tuple. If a different tuple is selected, the next attribute value will be retrieved from the new selected tuple.

#### Update Protocol

During the update action, attributes are individually updated one by one on the current selected tuple. The protocol may or may not be triggered by invoking a specific method. It ends when a specific method is invoked to commit the updated attributes. If another tuple or protocol (except the read protocol) is selected while it is being executed, all previous changes will be discarded.

#### Insert Protocol

The insert protocol is triggered by invoking a specific method. Then, each attribute is individually inserted one by one. After all attributes have been inserted, the protocol ends when a specific method is invoked to commit the inserted tuple. If another tuple or protocol (except the read protocol) is selected while it is being executed, all previous changes will be discarded.

#### Delete Protocol

The delete protocol comprises a single method that removes the current selected tuple from the LMS. The delete action is also committed in accordance with the established policy.

Table 2 presents the main protocols of LMS and the logic associated with each one.

ID	Protocol	Id	Protocol
1	Point to a tuple Read attributes	2	Point to a tuple Start update protocol Update attributes Commit update
3	Start insert protocol Insert attributes Commit insert	4	Point to a tuple Delete tuple

Table 2. Main protocols of LMS.

### 2.2.2.4 Access Modes of CLI to RDBMS

CLI are used, among other purposes, to access data residing in RDBMS. From the descriptions previously presented in 2.2.2.2 and 2.2.2.3, it is possible to infer that CLI provide two different modes to access data residing in RDBMS, which are herein referred to as the Direct Access Mode and the Indirect Access Mode.

#### Direct Access Mode

The Direct Access Mode is useful when programmers use native SQL to write CRUD expressions encoded inside strings and then delegate the remaining process to CLI to execute them against the RDBMS. CRUD expressions are of any type (Insert, Read, Update and Delete).

#### Indirect Access Mode

The Indirect Access Mode is only available after the execution of a Select expression (using the Direct Access Mode). CLI instantiate LMS and provide protocols (read, update, insert and delete) through which programmers are allowed to interact with the in-memory data of LMS. These protocols belong and constitute the Indirect Access Mode. Whenever an update, insert or delete protocol is committed, CLI internally create the correspondent CRUD expression to commit the changes. The read protocol is also included in the Indirect Access Mode.

Current approaches to enforce access control are only based on the Direct Access Mode. But, unlike current approaches, the DACA implements FGACM at the level of CLI and, as such, it must implement FGACM not only on the Direct Access Mode but also on the Indirect Access Mode for all

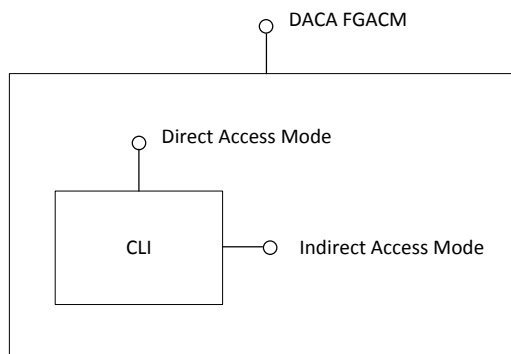


Figure 9. CLI and DACA access mechanisms.

protocols (read, update, insert and delete). Figure 9 shows a simplified block diagram to slightly unveil the approach to be followed by the DACA to implement FGACM on CLI. The diagram shows that the two access modes of CLI are wrapped and hidden by the DACA to support data access ruled by FGACM only.

### 2.2.2.5 Performance

Very often, performance is a key requirement and frequently a bottleneck of database applications when scalability limitations arise. As such, an overview about the issues that influence performance is an unavoidable aspect of CLI and it must comprise the two described access modes of CLI.

#### Direct Access Mode

Native CRUD expressions are edited and executed through the Direct Access Mode. The execution context has two mutual-exclusive possibilities: CRUD expressions are compiled-on-the-fly or CRUD expressions are pre-compiled. As already explained the pre-compiled approach compiles and stores CRUD expressions on the RDBMS. Whenever needed, CRUD expressions are already compiled and their execution plans have been already evaluated. The performance improvement is mostly noticeable when CRUD expressions use complex execution plans.

#### Indirect Access Mode

The Indirect Access Mode is available whenever a Select expression is executed. When a Select expression is executed using a scrollable or an updatable LMS, RDBMS create a server cursor with all the selected tuples. These tuples are dynamically transferred in blocks, from the server, to the LMS whenever necessary. This means that at any time LMS may not have all the tuples but only a sub-set of all selected tuples. When users point to a tuple that is not present in the LMS, the current content of LMS is discarded and a new set of tuples containing the desired tuple is transferred to the LMS. This has a deep implication. If threads are always requesting tuples that are not present in the LMS, RDBMS have to transfer the correspondent block for each request. In an extreme scenario, each individual action over the LMS could imply a new transference of tuples. From the previous statements, it is expected that the number of blocks to be transferred will increase when the number of tuples (inside server cursors) increases and also when the dispersion of the used policy to select tuples (contained by LMS) increases. Thus, to optimize the performance two strategies need to be followed [Pereira, '10b; Pereira, '11b; Pereira, '13c]. The first one and simplest one is to avoid the use of scrollable and/or updatable LMS. If it is not possible to avoid the use of scrollable or updatable LMS, then access to LMS should follow a policy aimed at minimizing the transferences of block of tuples.

### 2.2.3 Other proposals

Beyond O/RM and CLI, several other tools have been launched by the research community and the commercial community. In his sub-section an overview of the most relevant tools is made.

Embedded SQL [Moore, '91] is a method for writing CRUD expressions in-line with regular source code of the host programming language inside source files. The CRUD expressions provide the database interface while the host programming language provides the remaining support needed for the application to execute. The files are then pre-processed (pre-compiled) in order to

check the correctness of CRUD expressions namely against the database schema, host language data type and SQL data type checking, and finally syntax checking of the SQL constructions. SQLJ [Eisenberg, '99] is an example of an Embedded SQL standard, which provides language extensions for embedding CRUD expressions in regular Java source files. Some SQLJ disadvantages, which are common to most Embedded SQL technologies: 1) SQLJ relies on an extra standard; 2) SQLJ does not decouple CRUD expressions from regular source code; 3) SQLJ does not provide a clean object-oriented interface to the assisted application; 4) SQLJ does not provide assistance regarding the maintenance of CRUD expressions; 5) SQLJ requires a JVM (Java Virtual Machine) built in the database. In practice, embedded SQL has never been widely adopted by end users. Examples of other languages that support embedded SQL are: C, C++, COBOL and Fortran. Despite the aforementioned general disadvantages, some embedded SQL features may be considered as advantages such as: it is based on a single development environment with a strong interconnection between the two paradigms; unlike other solutions, embedded SQL does not need to be executed to check the correctness of the SQL syntax. This task is executed by the pre-compiler.

Safe Query Objects [William, '05] combine object-relational mapping with object-oriented languages to specify queries using strongly-typed objects and methods. They rely on Java Data Objects to provide strongly-typed objects and also to provide data persistence. Safe Query Objects are a promising technique to express queries but share most of the aforementioned drawbacks of O/RM, namely regarding performance and SQL expressiveness.

SQL DOM [Russell, '05] generates a Dynamic Link Library containing classes that are strongly-typed to a database schema. These classes are used to construct dynamic CRUD expressions without manipulating any strings. As Safe Query Objects, SQL DOM does not take the full advantage of SQL expressiveness and also exhibits very poor results regarding its performance.

Static Checking of Dynamically Generated Queries [Gary, '07] presents a solution based on static string analysis of Java programs to find out where CRUD expressions are being constructed. The main idea is to find out all possible combinations of distinct CRUD expressions and then analyze them regarding their syntax and their type mismatch errors. This approach does not affect system performance but exhibits some drawbacks as: 1) all source code is hand written from string concatenation till JDBC execution context; 2) it does not provide any object-oriented view of the CRUD expression execution context.

In [Schmoelzer, '06] Schmoelzer et al. do not present a tool but present a concept for model-typed interfaces relying on generic interface parameters that may be used to transfer data. The parameters are characterized as Model-defined Types whose schema is defined by a Data Model. The authors claim that by this way, complex data structures (based on Data Models) may be transferred between components in a single method invocation avoiding successive calls to accomplish the same task. This methodology is very useful when two conditions occur simultaneously: 1) the involved components do not share the same working address space; 2) the component playing the client role has full control and knowledge about the amount of data being transferred. In our case, Business tiers based on the DACA and client applications share the same address space. Then, the best access method to the returned data (from Select expressions) is implemented in attribute by attribute and tuple by tuple basis. The DACA could profit from [Schmoelzer, '06] if systems based on the DACA and client applications ran in different address spaces.

Data Transfer Objects [Flower, '02] is another concept for the transference of data. It proposes a design pattern to be used whenever an entity gathers a group of attributes that must be accessed in a swift way. Accessing those attributes one by one through a remote interface raises several



disadvantages such as the increase of the network traffic, latency is increased, performance is negatively affected, demand on server and client processing is increased. Data Transfer Objects are tailored to address these situations. They are organized in serializable classes gathering the related attributes and forming a composite value. An entire instance of the serialized object is transferred from the server to the client. This approach, in its essence, is quite similar to the previous conveying the same advantages and disadvantages.

Aspect-oriented programming [Gregor Kiczales, '97] community considers persistence as a crosscutting concern [Laddad, '03]. Several works have been presented but none addresses the point here under consideration. The following works are emphasized: [Fabry, '06] is focused on separating scattered and tangled code in advanced transaction management; [Laddad, '03] addresses persistence relying on AspectJ; [Dinkelaker, '11] presents AO4Sql as an aspect-oriented extension for SQL aimed at addressing logging, profiling and runtime schema evolution. It would be interesting to see an aspect-oriented approach for the points herein under discussion.

### 2.3 JDBC

JDBC is the CLI used in the DACA proof of concept. To provide the necessary background to completely understand the DACA, JDBC is now presented and described as the representative of CLI.

#### 2.3.1 JDBC Overview

JDBC is a CLI version for a standard Java specification for database-independent connectivity. There are four styles of drivers, see Figure 10:

- JDBC-ODBC Bridge plus ODBC Driver – type 1
- native API partly Java technology-enabled driver – type 2
- Pure Java driver for database middleware – type 3;
- Direct-to-database pure Java driver – type 4.

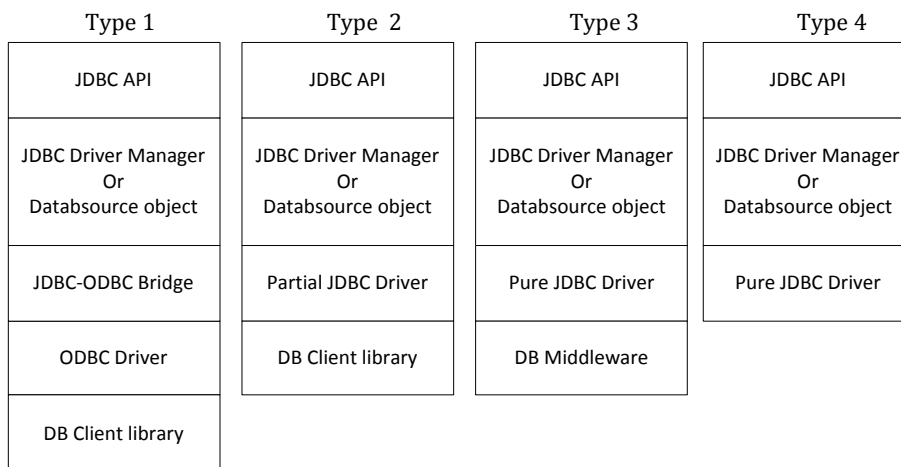


Figure 10. Types of JDBC drivers and their dependency on other components.

JDBC Type 1 driver uses ODBC driver to connect to the database. A database client library is also necessary if ODBC is not a native driver to the RDBMS.

JDBC Type 2 driver converts JDBC calls into calls on the client side vendor's API to connect to the database.

JDBC Type 3 driver converts JDBC calls directly or indirectly into the middleware client side libraries of the database.

JDBC Type 4 driver converts JDBC calls into the network protocol used to connect to the database and, as such, is considered the best choice when performance is considered a key requirement. The proof of concept of DACA uses a Type 4 driver for SQL Server: *sqljdbc4.jar*.

### 2.3.2 JDBC Approach to Call Level Interfaces Functionalities

As previously explained, main functionalities of CLI are organized in four main categories: execution, scrollability, updatability and transactions. Figure 11, Figure 12, Figure 13, Figure 14, Figure 15 and Figure 16 present typical JDBC usage of the four main functionalities. These figures will be referred during the next explanations. Figure 11 presents the declaration of the main variables used in these examples: *Statement* is an object aimed at executing compiled-on-the-fly CRUD expressions, *PreparedStatement* is an object aimed at executing pre-compiled CRUD expressions and *ResultSet* is an object responsible for managing LMS. The line numbers in all figures are not repeated between them, thus, whenever dispensable we will not refer the figures being used in this sub-section.

#### Execution

Execution comprises services related to the execution of CRUD expressions. JDBC uses *PreparedStatement* [Oracle, '12b] and *Statement* [Oracle, '12c] for pre-compiled and compiled-on-the-fly SQL statements, respectively.

#### Pre-compiled CRUD expressions

Figure 13 and Figure 15 show the usage of pre-compiled CRUD expressions (*PreparedStatement ps*). CRUD expressions are written (line 49, 84) and compiled (line 50-52, 85). This is done only once and then CRUD expressions are re-executed whenever necessary (line 56, 90).

#### CRUD expressions compiled-on-the-fly

Figure 12 and Figure 14 show the usage of compiled-on-the-fly CRUD expressions (*Statement st*). CRUD expressions are written (line 36, 66), then the context is prepared (line 37, 67) and finally CRUD expressions are executed (line 39, 69). This process is repeated from the very beginning whenever any of the CRUD expressions is required to be executed.

```
31 private Statement st;  
32 private PreparedStatement ps;  
33 private ResultSet rs;
```

Figure 11. Declaration of variables.

```

35 private void readSt(int productId) throws SQLException {
36     sql="Select * from Products p Where p.ProductId=" + productId;
37     st=conn.createStatement(ResultSet.TYPE_FORWARD_ONLY,
38                             ResultSet.CONCUR_READ_ONLY);
39     rs=st.executeQuery(sql);
40     if (rs.next()) {
41         productName=rs.getString("productName");
42         supplierId=rs.getInt("supplierId");
43         categoryId=rs.getInt("categoryId");
44         // read other attributes
45     }
46 }

```

Figure 12. Use of forward-only and read-only statement.

```

48 private void prepareStatement_foRo() throws SQLException {
49     sql="Select * from Products p Where p.ProductId=?";
50     ps=conn.prepareStatement(sql,
51                             ResultSet.TYPE_FORWARD_ONLY,
52                             ResultSet.CONCUR_READ_ONLY);
53 }
54 private void readPS(int productId) throws SQLException {
55     ps.setInt(1, productId);
56     rs=ps.executeQuery();
57     if (rs.next()) {
58         productName=rs.getString("productName");
59         supplierId=rs.getInt("supplierId");
60         categoryId=rs.getInt("categoryId");
61         // read other attributes
62     }
63 }

```

Figure 13. Use of forward-only and read-only prepared statement.

```

65 private void readSt(List<Integer> id, List<Integer> us) throws SQLException {
66     sql="Select * from Products p";
67     st=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
68                             ResultSet.CONCUR_UPDATABLE);
69     rs=st.executeQuery(sql);
70     while (rs.next()) {
71         productId=rs.getInt("productId");
72         int idx=id.indexOf(productId);
73         if (idx!=-1) {
74             unitsInStock=us.get(idx);
75             rs.updateInt("unitsInStock",unitsInStock);
76             rs.updateRow();
77         }
78     }
79     rs.beforeFirst();
80     // more code
81 }

```

Figure 14. Use of scrollable and updatable statement.

```

83 private void prepareStatement_insert() throws SQLException {
84     sql="insert into products values (?, ?, ?, ?, ?, ?, ?, ?, ?)";
85     ps=conn.prepareStatement(sql);
86 }
87 private int insert(Object[] values) throws SQLException {
88     ps.setString(1, (String)values[0]);
89     // ... more attributes
90     int n=ps.executeUpdate();
91     return n;
92 }
    
```

Figure 15. Insert a row using a prepared statement.

```

109 private void transaction() throws SQLException {
110     conn.setAutoCommit(false);
111     try {
112         //... execute some CRUD expressions
113         conn.commit();
114     } catch( SQLException ex ) {
115         conn.rollback();
116     }
117     conn.setAutoCommit(true);
118 }
    
```

Figure 16. Examples of transaction with JDBC.

Additionally, CLI deal differently with Select expressions from the other three types of CRUD expressions. Select statements instantiate an LMS (line 39, 56, 69), while the other types do not. These latter types generate a value indicating the number of affected rows in the database (line 90).

**Scrollability**

Scrollability comprises services related to the scrolling process on LMS. There are two mutual-exclusive possibilities: *forward-only* (line 37, 51) – in this case it is only possible to move forward one row at a time, (line 40, 57); *scrollable* (line 67) – in this case it is possible to move in any direction and jump several rows at a time (line 79). There are several other methods as shown in Figure 17. Additional detail can be found in [Oracle, '12c].

«interface» <b>ForwardOnly</b>	«interface» <b>Scrollable</b>
+isAfterLast() : bool	+absolute(in position : int) : bool
+isBeforeFirst() : bool	+afterLast()
+isFirst() : bool	+beforeFirst()
+isLast() : bool	+first() : bool
+next() : bool	+isAfterLast() : bool
	+isBeforeFirst() : bool
	+isFirst() : bool
	+isLast() : bool
	+last() : bool
	+next() : bool
	+previous() : bool
	+relative(in rows : int) : bool

Figure 17. Methods to scroll on LMS.

### Updatability

Updatability comprises services organized in protocols to interact with data contained in LMS. There are two mutual-exclusive possibilities: *read-only* (line 38, 52) – the content of the LMS is read-only and no changes are allowed; *updatable* (line 68) – changes may be performed on LMS (insert new rows, update rows (line 75-76) and delete rows). CLI commit these changes into the host database. The important aspect in these actions is that CLI, internally, create CRUD expressions to execute the actions performed at the LMS level. Thus, when line 76 is executed JDBC creates an Update expression to update the modified attribute. Similarly, when a tuple is inserted or deleted at the LMS level, JDBC creates Insert and Delete expressions to materialize the requested actions.

### Transactions

Transactions comprise a set of services to manage database transactions such as save points, rollbacks and commits. Figure 16 presents a scenario where the auto-commit mode is changed into the normal mode (line 110), some CRUD expressions are executed (line 112) and committed (line 113). If an SQLException is caught, the transaction is rolled back (line 115). Finally, the auto-commit mode is replaced (line 117).

## 2.3.3 JDBC Class Diagram

Main functionalities of JDBC are organized around four interfaces: Connection [Oracle, '12a], Statement [Oracle, '12c], PreparedStatement [Oracle, '12b] and ResultSet [Oracle, '13] as shown in Figure 18.

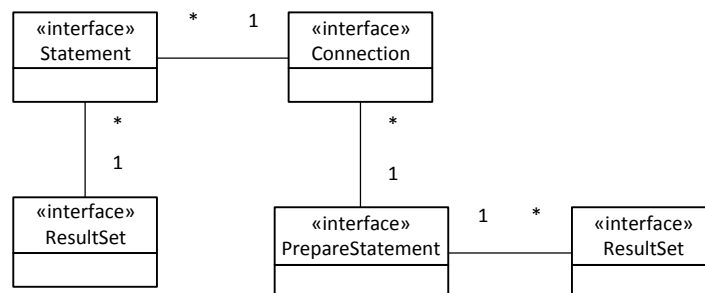


Figure 18. JDBC class diagram.

### Connection

The root interface is the Connection interface which manages a connection to a database.

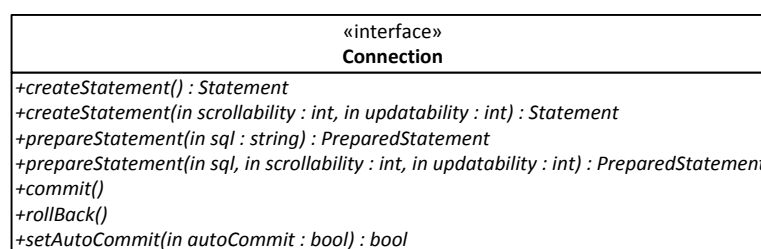


Figure 19. Connection interface.

Among others, it provides services for transactions management and for creating Statement and PreparedStatement objects, see Figure 19. Additional detail about the Connection interface is here provided [Oracle, '12a]

### Statement

The Statement interface manages the execution of compiled-on-the-fly CRUD expressions, see Figure 20. Among others, it provides the following two main services:

- `executeQuery`: to execute Select expressions, which returns a *ResultSet* (LMS in JDBC);
- `executeUpdate`: to execute Update, Insert and Delete expressions, which returns an integer to indicate the number of affected rows.

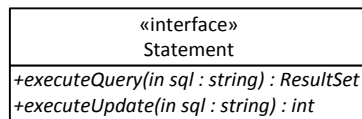


Figure 20. Statement interface.

Additional details about the Statement interface can be found in [Oracle, '12c].

### PreparedStatement

The PreparedStatement interface manages the execution of pre-compiled CRUD expressions, see Figure 21. Among others, it provides the following services:

- `executeQuery` and `executeUpdate`: same as in Statement interface;
- others: the remaining services are used to set the runtime values for the parameters of CRUD expressions. There is one method for each data type.

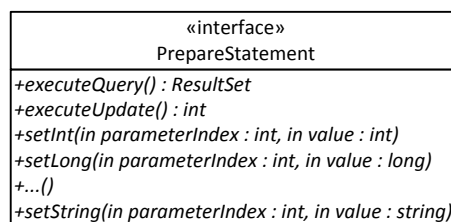


Figure 21. PreparedStatement interface.

Additional details about the PreparedStatement interface is here provided [Oracle, '12b].

### ResultSet

The ResultSet interface is the JDBC implementation of LMS, see Figure 22. ResultSet is a plane interface comprising all services independently from its instantiation context: forward only or scrollable and, read-only or updatable. This means that programmers must remember the context in which a ResultSet was instantiated to only use the valid and active services. Otherwise, exceptions will be raised. One important aspect, as it will be shown, is that programmers need to master the schema of the returned relation to be able to access the data

managed by `ResultSet`. This does not happen with O/RM tools, such as Hibernate, JPA and LINQ. This is indeed a drawback of CLI regarding their usability. The DACA overcomes this CLI drawback by providing type safe and database schema-driven methods. Among others, `ResultSet` interface provides the following methods, as shown in Figure 22:

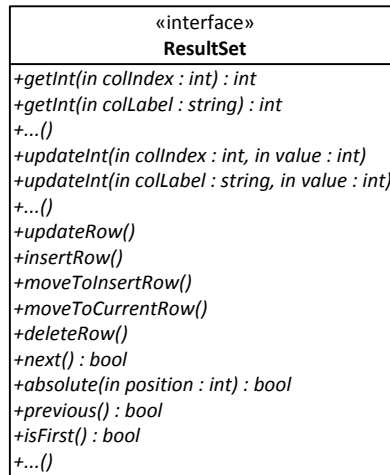


Figure 22. `ResultSet` interface.

### getInt

The `getInt` method is used to read data of type integer returned by Select expressions. There are two methods for each data type. One method uses the column index and the other uses the column label (example - Figure 12: line 42, 43) to read the data. Only the methods for the integer data type are here presented and described. As previously mentioned, programmers need to master the schema of the returned relation.

### updateInt

The `updateInt` method is used to update columns of data type *integer* and also to insert values in columns of data type *integer*. There are two methods for each data type. One method uses the column index and the other uses the column label (example - Figure 14: line 75). Only the methods for the integer data type are here presented. As previously mentioned, programmers need to master the schema of the returned relation.

### updateRow

The `updateRow` method is used to confirm previously updated values; if operating in auto-commit mode, the updated columns are committed.

### insertRow

The `insertRow` method is used to confirm previous inserted values; if operating in auto-commit mode, the updated columns are committed.

### moveToInsertRow

The `moveToInsertRow` method is used to set the cursor used by `ResultSet` to point to the

tuple where new tuples are inserted. Thus, this method has to be invoked to start the insert protocol.

#### moveToCurrentRow

The moveToCurrentRow method is used to restore set the previous cursor position (before moveToInsertRow).

#### deleteRow

The deleteRow method deletes the current selected tuple; if operating in auto-commit mode, the deleted tuples are committed;

#### remaining methods

The remaining methods are used to scroll on the ResultSet to select one of the tuples.

Additional detail about the ResultSet interface can be found in [Oracle, '13].

## **2.4 Current Approaches to Implement Access Control**

In the context of RDBMS, access control is focused on protecting sensitive data managed by RDBMS from legitimate users. Legitimate users are entities identified with username and password to access databases. In order to control the access to data, several approaches have been presented each one with its own characteristics and goals. The diversity of the approaches does not ease their classification in major groups even if the classification follows the previous presented and described dimensions of access control: architecture, granularity, awareability, contextuality and adaptability. As such, the presentation of current approaches is split in two sub-sections. In the first sub-section current approaches are organized and presented by technological aspects and in the second sub-section the major research approaches are individually described and presented, and not grouped under any classification.

As far as we know, no previous work has addressed the key aspects of this thesis. Two of those key aspects are 1) the implementation of FGACM at the business tier level, built at runtime, and kept updated when policies evolve and 2) business tiers driven by FGACM and based and exploiting CLI features such as their multi-access mode: Direct Access Mode and Indirect Access Mode. The only aspect that has been addressed by some researches is awareness about the established FGACP.

### **2.4.1 Current Techniques**

Several techniques have been devised and used to implement access control. This sub-section presents and describes some of the most used technical solutions for access control.

#### **Views**

Database views may be used as the basis for FGAC. Views are standard database entities that aggregate selected and filtered data. Then, these data are used to evaluate the disclosing policy to legitimate users in accordance with the established FGACP. Next follows an example of a view restricting the access to rows of a table with ids < 100.



```
Create view myTable as
  Select * from table
  Where id < 100
```

From now on, authorization is evaluated against the view and not against the original table. The most usual implementations prohibits users from issuing the CRUD expression

```
Select * from table where id < 100
```

and, instead, they issue the following CRUD expression

```
Select * from myTable
```

Albeit being an easy technique, the use of views presents some disadvantages. For example, the use of views is not scalable. The number of views increases with the number of policies. Moreover, users accessing the same table but with different authorizations need different views. While from the database schema point of view this means an unbounded number of views, from the business tier point of view this means an unbounded number of CRUD expressions. These disadvantages may be unsustainable in large databases with complex schemas and many and complex access control policies. In order to minimize this scalability drawback, [Rizvi, '04] proposes an approach where users always issue CRUD expressions against the original tables but the authorizations are evaluated against security views.

Anyway, views also present one significant advantage. Views are relational entities supported by the standard SQL language this way avoiding the need to additional tools or additional techniques. As relational entities, they are kept together with database tables this way conveying a single point for their development, deployment and maintenance.

### **Parameterized Views**

A parameterized view is an SQL view definition which makes use of runtime parameters like *user-id*, *time* and *user-location*. The next example shows a simple parameterized view.

```
Create view myTable as
  Select * from table
  Where id=SYSTEM_USER
```

This parameterized view lets the legitimate user to see all rows from table *Table* where the *id* matches his/her user identification. Parameterized views are used to create different authorization accesses based on a single view and a single CRUD expression, which is a different and more efficient than the traditional views just presented. Thus, parameterized views convey the same advantages and disadvantages as traditional views, but positively contribute to minimize the lack of scalability of traditional views.

Parameterized views is also the approach proposed in [Roichman, '07] to implement FGAC in Web databases. Basically, each user is identified as belonging to a group to which a set of parameterized views is assigned.

### **Query Rewriting**

Query rewriting is a technique used to rewrite CRUD expressions before their execution to avoid unauthorized access to protected data. The rewriting process is usually conducted in a central server and several techniques have been proposed. Next follows the presentation of some of the most used techniques.

#### **Addition of predicates**

Appending predicates to *where* clauses is one of the used techniques to rewrite CRUD expressions. Predicates are used to filter the data to be disclosed, in accordance with the established FGACP. For example, the CRUD expression

```
Select * from table
```

is replaced by the CRUD expression at runtime

```
Select * from table
      Where (some condition)
```

#### **Tables replaced by views**

This technique is used to replace names of tables by names of views representing the authorized data. For example, the CRUD expression

```
Select * from table
```

is replaced by

```
Select * from myTable
```

where *myTable* is a view of table *Table* containing the authorized data.

#### **Masking cells**

Masking cells technique rewrites CRUD expressions to mask protected data that is returned by Select expressions. For example, the CRUD expression

```
Select column from ...
```

is replaced by the CRUD expression

```
Select column = CASE somePolicy
      When hide
      Then HiddedValue
      Else column
```

The rewritten CRUD expression uses at runtime a policy, *somePolicy*, to decide upon disclosing attitude for the protected column. There are two main approaches to mask cells. One uses named variables to represent protected data. The other uses the standard SQL NULL values. Named variable are by far the best choice but named variables are not

supported by all RDBMS. The NULL value is the alternative and it is easily implemented. The drawback is that the use of NULL values to protect data prevents the distinction between real SQL NULL values and a hidden protected value.

#### Column removal

The column removal technique removes all protected columns from the select list (projected attributes). For example, if the data contained in column *col\_B* of CRUD expression

```
Select col_A, col_B, col_C from table
```

is not authorized to be disclosed, the CRUD expression is rewritten as

```
Select col_A, col_C from table
```

This technique effectively hides the protected data but exceptions are raised if client-side applications try to use the hidden column *col\_B*. Moreover, the same CRUD expression when used by users with different authorizations returns relations with different schemas. This situation inevitably raises several difficulties not only during the development process of client-applications but also during maintenance activities, which are both significantly hampered.

Query rewriting technique has advantages and disadvantages. Among them, the following advantages are emphasized:

#### Transparency

Query rewriting has the advantage of being transparent to database users. Database users write CRUD expressions as if no security policy is implemented. Then, CRUD expressions are rewritten in accordance with the established policies.

#### Scalability

From application tiers point of view there is no need to extend the number of CRUD expressions to conform with the established FGACP. CRUD expressions are written as if no policy was defined and then, at runtime, they are automatically rewritten.

Thus, query rewriting overcomes the main disadvantages of views and parameterized views. Meanwhile, query rewriting conveys some disadvantages and threats. Among them the following are emphasized:

#### Unawareness

The query rewriting process is an independent process out of the scope of database users. Queries violating any security policy are rejected and users are pushed to deal with corrective activities, very often with no feedback about the causes of the rejection.

#### Performance decay

The query rewriting process is usually processed by a centralized system and requires a certain amount of computational resources. In spite of not being mandatory, in most of the

cases the systems responsible for the query rewriting process are the RDBMS themselves. Additionally, after being rewritten, the performance of original CRUD expressions very probably has decay in performance.

#### Dead lock

Dead locks may occur if policies use data from the tables being queried. These undesirable situations are caused by non-terminating loops when policies recursively invoke themselves when the table is queried. To prevent such dead locks additional expressive power is needed [Olson, '08].

The query rewriting technique is widely used and proposed as solution to address FGACP. Among them, the following proposals are emphasized: Oracle [Oracle], [LeFevre, '04], [Rizvi, '04], [Wang, '07] and [Barker, '08].

#### **Extensions to SQL**

Currently, the standard SQL only permits limited forms of access control. Some of the forms are the GRANT, REVOKE and DENY commands. These commands are far from coping with current security needs. Extensions to the standard SQL have been proposed by several authors to tackle the current security gap of the standard SQL language. Some contributions have been proposed to extend the SQL standard, such as in [Chlipala, '10] through the *known* predicate and [Chaudhuri, '07] by the generalization for the current SQL authorization mechanisms. Even if the SQL standard was extended to deal with all security requirements, it would rely in a centralized architecture conveying all the described advantages and disadvantages.

#### **Language extensions, security languages and tools**

Language extensions, security programming languages and tools have been proposed to address FGACP. Current programming languages are extended and specialized functionalities are included to address access control. Several researches have been conducted in this direction. Among them the following are emphasized: SELINKS [Corcoran, '09] extends LINKS to build secure multi-tier web applications; Jif [Zhang, '12] is a Java extension which uses labels in-line with the source code to express access control policies; [Fischer, '09] introduces objects-sensitive types driven by RBAC policies to overcome Java EE *@RolesAllowed* annotation approach to RBAC. New programming languages have also been devised. In [Caires, '11], Caires et al. present a new programming language named as  $\lambda_{DB}$  for verifying and for expressing FGACP. In [Ribeiro, '01], Ribeiro et al. present a security programming language aimed at integrating heterogeneous security policies. Some tools have also been devised. In [Chlipala, '10], Chlipala et al. present a tool, *Ur/Web*, that allows programmers to write statically-checkable FGACP as SQL queries.

#### **PEP-PDP**

Solutions based on PEP-PDP approach are based on the mixed architecture. Basically, PEPs are included in-line with the source code of client applications to enforce the policies decided by a PDP placed in a remote server. If authorization is granted, PEP executes the requested action otherwise the requested action is refused. The best well-known proposal is the XACML [OASIS,

'12] standard from OASIS but other research proposals have also been presented and based on this architecture. SELINKS [Corcoran, '09] has also proposed a PEP-PDP approach for multi-tier web database applications.

### **Semantic Access Control**

Semantic access control (SAC) uses Semantic Web [Berners-Lee, '01] concepts to the access control area. The main difference to the remainder approaches is that decisions are based on the semantic of attributes, such as resources and users, and not on stored or hard coded information. It has been used in several domains such as: to integrate access control between heterogeneous data repositories [Hu, '11; Pan, '06; Warner, '07], to provide secure content access and distribution [Lopez, '02a; Valle, '02] and to extend semantic web concepts to RBAC models [Ao, '04; Kim, '10].

### **RDBMS Vendors**

Access control has been a permanent worry of RDBMS vendors. RDBMS vendors have been providing embedded tools from which security experts build and maintain access control to the data to be protected. Granularity of access control in RDBMS started to be at the database object (tables and views) level. This granularity became inadequate when the claim for more security increased. To cope with this increased demanding, RDBMS vendors started to support finer-grained access control. Different approaches were followed. INGRES and Oracle uses a query rewriting technique while DB2 [Bond, '07] uses a label-based technique. In spite of the diversity of policies, RDBMS vendors have elected the RBAC as the preferred choice. Each RDBMS vendor provides its proprietary approach leading to a situation where access control is far from being standardized. Very often the security features of RDBMS are not enough and there is the need to provide a different approach to access control. This need has been the motivation for the researches that have been conducted and also one of the motivations of this thesis. A radical approach is the one provided by Hippocratic databases [Agrawal, '02; LeFevre, '04].

## **2.4.2 Related Work**

In this sub-section follows the presentation of work related with the enforcement of ACP (Access Control Policies) and FGACP.

### **Virtual Private Database**

Oracle addressed FGACP by introducing the *Virtual Private Database* [Oracle] technology. This technology is based on rewriting CRUD expressions before their execution and in accordance with the established FGACP. The authorization policy is encoded into functions defined for each relation, which are used to return *where* clauses predicates to be appended to CRUD expressions to limit data access at the row level. This approach provides a per-user view of each database object (called Truman model in [Rizvi, '04]). Next follows an example based on two tables:

```
Doc_Doctor {doc_id doc_name,...}
```

Pat\_Patient {Pat\_id,PatDoc\_id,Pat\_name,...}.

If doctors are restricted to see only their patients, if the following CRUD expression is issued by a doctor

```
Select * from Pat_Patient
```

it will be automatically rewritten to

```
Select * from Pat_Patient where PatDoc_id=<id of the doctor logged in>
```

To set-up the access control, a function is written to compute the predicate to be added to the CRUD expression and a policy is placed on the table Pat\_Patient. The function needs to select Doc\_id from Doc\_Doctor for the doctor logged in and then constructs the predicate automatically.

It is also possible to use Virtual Private Database at the column level to prevent disclosure of protected data. There are two alternatives: column removal (default behavior) - all cells containing sensitive data are removed; cell masking - content of cells containing sensitive data is replaced by NULL value.

Virtual Privacy Database is an alternative to views by avoiding some of their drawbacks such as the need for an additional view for each policy. With the Virtual Private Database technique, the same CRUD expression is shared by all users and automatically modified in accordance with the permission of each user.

### **Hippocratic databases**

In 1974 the United States Privacy Act defined a set of rules for limiting the collection, use and dissemination of personal data held by Federal Agencies [Agrawal, '02]. The defined concepts are generally known as Fair Information Practices [Systems, '73] and have been used to develop important international guidelines for privacy protection [Agrawal, '02]. From these guidelines, [Agrawal, '02] announces ten principles to characterize Hippocratic databases. Hippocratic databases aim at integrating privacy policies into database architectures. The ten principles are: purpose specification (for which the information has been collected), consent (purpose must be consented by the donor), limited collection (minimum necessary for accomplishing the specified purpose), limited use (run only those queries that are consistent with the purpose for which the information has been collected), limited disclosure (information shall not be communicated outside the RDBMS for purposes other than those consented), limited retention (only until the fulfillment of the purpose), accuracy (information must be accurate and up-to-date), safety (information must be protected by security mechanisms), openness (a donor is able to access to its own information) and compliance (a donor is able to verify compliance with the principles) [Agrawal, '02; Kirchberg, '10; LeFevre, '04]. Some efforts have been made to bring those principles into practice, among which the ones of IBM [IBM, '07] and PostgreSQL [Padma, '09] are emphasized. The following example is based on Hippocratic PostgreSQL

```
Select p.productName, p.unitsInStock, p.unitsOnOrder, p.reorderLevel  
From Products p
```

Purpose stockControl  
Recipient stockManager

The result of this Select will be restricted to include only the columns that the combination of purpose and recipient is allowed to access according to the policy specification. It will be further restricted to include only data of products to be shared with stockControl. From this simple example it is clearly seen that databases addressing Hippocratic principles diverge from traditional RDBMS. Additionally, Hippocratic databases also address data privacy which is a distinct form of access control. While privacy is concerned with the right of individuals to determine for themselves when, how and to what extent information about them is communicated to others, access control is concerned with controlling which legitimate users are allowed to access protected data.

#### **[LeFevre, '04]**

In [LeFevre, '04] LeFevre et al. propose a technique to control the disclosing data process in Hippocratic databases. The disclosing process is based on the premise that the subject has control over who is allowed to see its protected data and for what purpose. It is based on the query rewriting technique. Policies are defined using P3P [W3C, '02] or EPAL [W3C, '03] and comprise a set of rules that describe to whom the data may be disclosed and how the data may be used. Two disclosure models are supported for cells: at the table level - each purpose-recipient pair is assigned a view over each table in the database and prohibited cells are replaced with null values; at the CRUD expressions level - protected data are removed from the returned relations of Select expressions, in accordance with the purpose-recipient constraints. Rules are stored as meta-data in the database. CRUD expressions must be associated with a purpose and a recipient, and are rewritten to reflect the ACP.

#### **SESAME [Zhang, '03]**

SESAME [Zhang, '03] is a dynamic context-aware access control mechanism for pervasive GRID applications. It relies on a dynamic role based access control model (DRBAC) which extends the classic RBAC model. Basically, DRBAC assigns default role hierarchies when subjects log in. Afterwards, context of subjects are monitored and roles are dynamically delegated. SESAME and DRBAC model have been implemented as part of the Discover [Bhat, '03; Mann, '01] computational laboratory. Two types of context are considered: object context and subject context. Object context is concerned about things related to users such as user's location, time, local resource and link state. Subject context is concerned with things related to systems, such as the current load, availability and connectivity for a resource.

An experimental evaluation was carried out in the Discover [Mann, '01; Mann, '02] computational laboratory to measure the induced overheads. SESAME follows a traditional approach to enforce access control policies in a central system, conveying all the drawbacks previously presented.

#### **SELINKS [Corcoran, '09]**

SELINKS [Corcoran, '09] is a programming language in the type of LINQ and Ruby on Rails which extends LINKS [Cooper, '07] to build secure multi-tier web applications. LINKS aims to reduce the impedance mismatch between the three tiers. The programmer writes a single LINKS program and the compiler creates the byte-code for each tier and also for the security policies (coded as user-defined functions on RDBMS). Through a type system object named as Fable [Swamy, '08], 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. Programmers define security metadata (termed labels) using algebraic and structured types and then write enforcement policy functions that applications call explicitly to mediate the access to labeled data. Some of the security strengths of SELINKS are:

#### Security

SELINKS is a cross-tier security technique this way ensuring an integrated security context for the three tiers. Additionally, it uses Fable to ensure that security policies cannot be avoided, to ensure that security policies are correctly enforced and correctly called whenever a user tries to access protected data.

#### Integrated environment

SELINKS is cross-tier security technique relying on a single tool. This environment clearly eases the development process of database applications based on multi-tier architectures. Programmers do not need to master several tools and, above all, ensure their integration and coordination to reach a high level of security.

#### Optimized latency

User defined functions run on database servers and not on web servers, avoiding the overhead of needlessly transferring data between the web server and the database server.

#### Flexibility

Beyond access control, SELINKS allows other variety of security policies to be expressed: information flow [Denning, '76], provenance [Buneman, '06] and automaton-based policies.

Some of the security weaknesses of SELINKS are:

#### Additional technique

In spite of its advantage of relying on a single tool, programmers need to master a new tool, SELINKS, to develop secure applications.

#### Scope

Security labels specify a group-based access control policy, with separate access restrictions only for readers and writers of a record. There is no way to separate access restrictions by identifying inserts, updates and delete operations.

#### **Jif [Zhang, '12]**

Jif [Zhang, '12] is a security-typed programming language that extends Java with support for



information access control and also for information flow control [Denning, '76]. The access control is assured by adding labels in-line with the Java source code to express access control policies. The policy language supports: principals and labels, principal hierarchy, confidentiality and integrity constraints, robust declassification and endorsement and some language features such as polymorphism. 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 is not tailored to be an end-to-end access control tool to data residing in databases. It is mostly used to enforce security policies at the application level. As such, whenever used, JIF needs to be complemented with other techniques to manage the access control to data residing in RDBMS.

### **[Olson, '08]**

In [Olson, '08], Olson et al. describe a model for Reflective Database Access Control (RDBAC) based on the semantics of Transaction Datalog [Bonner, '97]. Privileges in the RDBAC model are expressed as CRUD expressions rather than as static privileges contained in access control lists. CRUD expressions use current state of databases to decide upon the accesses to be carried out. In [Olson, '09] a concrete implementation is provided. At the present moment, there is a model to define RDBAC policies and the CRUD expressions emanated from the policies. This result may be used as an input to the Direct Access Mode.

### **[Rizvi, '04]**

Rizvi et al. [Rizvi, '04] present a query rewriting technique to determine if a CRUD expression is authorized but without changing the CRUD expression. It uses security views to filter contents of tables and simultaneously to infer and check at runtime the appropriate authorization to execute any CRUD expression issued against the unfiltered table. The user is responsible to formulate the CRUD expression properly. They call this approach the Non-Truman model. Non-Truman models, unlike Truman models, do not change the original CRUD expression. The process is transparent for users and CRUD expressions are rejected if they do not have the appropriate authorization. The transparency of this technique is not always desirable particularly when it is important to understand why authorization is not granted so that programmers can revise their CRUD expressions more easily. This approach has some disadvantages:

#### Performance

The inference rules to check at runtime the appropriate authorization are complex and time consuming.

#### Productivity

Authorizations are checked against security views and not against original data. The process is transparent, so programmers do not know that their CRUD expressions are running against security views. If any syntax error or security violation occurs, the transparent process turns the debugging process more difficult.

### Awareness

Programmers cannot statically check the correctness of CRUD expressions because the policies and the mechanisms are centralized in a server. Programmers need to write, compile and run the source code to become aware of any security violation.

### Incompleteness

The inference rules are complex and their completeness is not assured by the authors.

### **[Dwork, '08]**

Differential-privacy [Dwork, '08] has had significant attention from the research community. It is mainly focused on preserving privacy from statistical databases. It really it does not directly address the point here under discussion. The interesting aspect is Frank McSherry's [McSherry, '10] 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.

### **[Morin, '10]**

Morin et al. [Morin, '10] use a security-driven model-based dynamic adaptation process to address access control and software evolution simultaneously. 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 focused on how to dynamically establish bindings between components from different layers to enforce security policies. They did not address the key issue of how to statically implement dynamic security mechanisms in software artifacts, in our case business tiers based on CLI.

### **[Roichman, '07]**

[Roichman, '07] argues that Web databases are particularly vulnerable to SQL injection attacks [Gregory, '05]. To overcome this security gap, authors propose an access control based on databases' built-in access control mechanisms: parameterized views [Eder, '96]. To address users' identification, a *Parameter method* is presented. Basically, users' identities are known (or automatically assigned using one of the proposed methods) and used to dynamically create parameterized views which gather the relevant data to the user, this way avoiding the access to unauthorized data. This approach is mainly focused on tackling SQL injection attacks and also on preventing users of Web databases to access databases without being previously identified. Users' identification may be considered a key aspect of access control but insufficient to address all aspects of access control. The authors themselves recognize that the proposed methodology is restrictive because it does not address every situations: "*The proposed access control mechanism is capable to prevent many kinds of attacks...*".

**[Chlipala, '10]**

[Chlipala, '10] presents a tool, *Ur/Web*, that allows programmers to write statically-checkable ACP as SQL queries. Basically, each policy determines which data is accessible. Then, programs are written and checked to assure that data involved in queries is accessible through some policy. To allow policies to vary by user, queries use actual data and a new extension to the standard SQL to capture '*which secrets the user knows*'. This extension is based on a predicate referred to as '*known*' used to model which information users are already aware of to decide upon the information to be disclosed.

Listing 1 presents a table *user* and its definition in *Ur/Web*. The policy expressed in Listing 2, named as *sendClient*, prevents users from reading data from other users. The predicate *known* models the information the user is already aware of. In this case, the user may read data about any row whose password he knows.

*Ur/Web* is a promising solution, but beyond introducing a new programming technique, it presents a key drawback of not checking access control to data of *where* clauses, allowing queries to implicitly leak protected data.

```
Table user: { Id: Int, Name: string, Pass: string }
```

Listing 1. Definition of table *user* in *Ur/Web*.

```
policy sendClient {
  Select *
  From user
  Where
  known(user.pass)
}
```

Listing 2. Policy definition in *Ur/Web*.

**[Caires, '11]**

[Caires, '11] presents a programming language, known as  $\lambda_{DB}$ , for expressing and verifying ACP by means of static type checking.  $\lambda_{DB}$  introduces programming structures known as *entities* which define database tables and the associated ACP. Then CRUD expressions are validated against the established ACP (at compile time) and also taking into account contextual information. Each permission is composed by:

- The granted action (either read or write);
- The list of attributes (entity fields);
- A condition expressed as a logical formula.

Listing 3 shows an entity named as *Person*. It comprises four attributes. Then ACP are defined for each attribute:

- *Public*: is readable in any condition as its associated condition (true) always hold;
- *Secret*: the content of this attribute in a row is readable only if the current user is the user identified in that row (*userid*) and the user is authenticated in the system;
- The write permission applies to all fields.

```
entity Person [userid: string; public: string; photo: picture; secret:
string]
  read public where true;
  ...
  read secret where Auth(uid) and uid=userid;
  ...
  write where Auth(userid);
```

Listing 3. Policy definition in  $\lambda_{DB}$ .

Beyond introducing a new programming technique this approach provides a unique action (write) to authorize update, insert and delete operations on attributes. This limitation clearly prevents a FGAC at the type of actions being executed. Unlike [Chlipala, '10], this approach provides access control to data of *where* clauses.

### **[Wang, '07]**

In [Wang, '07] three criteria are defined for enforcing FGACP. The algorithm should be *sound*, *secure* and *maximum*. “An algorithm is sound if the answer returned by it is consistent with the answer when there is no fine-grained access control policy. The algorithm is secure if the returned answer does not leak information not allowed by the policy. The algorithm is maximum if it returns as much information as possible, while satisfying the first two properties.” The rationale is presented and the work presented in [LeFevre, '04] is evaluated to conclude that it fails to satisfy the correctness criteria for FGACP. Authors use a labeling mechanism for cell-level disclosure policies to specify FGACP. Basically a policy determines whether a cell is viewable or not. This approach has also been used to work on privacy-centered database systems [Emilin Shyni, '10; LeFevre, '04]. Additionally, Wang argues that when one conceptual entity is split in two or more relations linked through foreign keys then the attributes involved in the linking process should be allowed even if the values of the keys cannot be released for privacy concern. In order to preserve useful information for query evaluation, two types of variables are defined to label unauthorized cells, this way avoiding the use of NULL value for protected cells. To prove the soundness of the algorithm, a query rewriting approach is presented to modify CRUD expressions in accordance with the established policies. Listing 4 presents a simple case to demonstrate the approach. The first CRUD expression is ruled by a policy where for each tuple in Employees, the value of attributes FirstName and HomePhone can be disclosed only when the disclosure conditions  $DC_{\text{FirstName}}$  and  $DC_{\text{HomePhone}}$  do not return 0 (zero), respectively. The rewritten CRUD expression employs the Case-Statement modification to mask unauthorized cells. Authors claim the soundness of the technique but some aspects need a further attention:

#### Applicability

The technique was applied to Select expressions. There is no evidence of its applicability to the three types of expressions: Insert, Update and Delete.

#### Performance

Authors conducted a performance evaluation with simple Select expressions and the

collected results suggest that scalability is compromised when the number of tuples is not small and efficiency is not a major concern.

```

-- original CRUD expression
Select FirstName, HomePhone
  from Employees

-- rewritten CRUD expression
Select FirstName =
      case DC_FirstName
        when 0
          then NULL
        else FirstName
      end,
      HomePhone =
      case DC_HomePhone
        when 0
          then NULL
        else HomePhone
      end
  From Employees

```

Listing 4. Query rewritten in T-SQL.

### **[Barker, '08]**

In [Barker, '08], Barker et al. provide support for representing, in SQL (DDL), dynamic fine-grained meta-level access control (DFMAC) policies. Meta-level policy is used to define different facts of ACP such as open and closed access control. DFMAC policies are presented as being important when goal-oriented access control requirements need to be represented. In goal-oriented access control, organizational and individual roles change as a consequence of the occurrence of events. Policies are represented in four tables:

- Category - to define to which categories users are assigned to;
- Policy - to store meta-level access control to be used by the query rewrite procedure;
- PCA - for permission category assignment and DCA - for denial category assignment.

From the data contained in these tables, and also from users' identification, queries are rewritten to enforce the established ACP.

This approach has the advantage of relying on SQL. Nevertheless, the work lacks of a deep performance evaluation because the presented examples suggest that the added predicates may have a significant impact on performance. Authors conducted some performance evaluations (not sufficiently described) and the collected results have shown an additional overhead of 10-15%, up to 26% and even "*pushed towards a bound of unacceptability*".

### **[Chaudhuri, '07]**

In [Chaudhuri, '07], Chaudhuri et al. propose a generalization for the current SQL authorization mechanism. The model is based on adding predicates to authorization grants and also on extending current SQL authorization model to support fine-grained authorization. Next follows a simple example

```
grant select on Employees
  where (employeeID=userId())
  to public
```

This authorization specifies that each employee is granted access to its own employee record. The model also supports nullification to control access at the cell level as shown in next example

```
grant select on Employees (address)
  where (some predicate)
  else nullify to public
```

This authorization specifies that access to the address attribute of Employees is granted only if the predicate is satisfied, otherwise a null value is returned.

The model also incorporates other features such as query for user groups and authorization groups to simplify administration activities.

The model addresses the following aspects:

- Predicates can be applied on any form of grant: CRUD expressions, functions and stored procedures;
- Nullification of values based on predicates to allow cell-level security [LeFevre, '04];
- Authorization on aggregates while limiting the access to raw data;
- Mechanisms to ease the administration of large number of application users.

To avoid large number of database users, the notion of user is defined at the application user level. As such, users of applications must be authenticated and their identity made available to the database.

### **Java EE**

Java EE supports the enforcement of RBAC policies through the `@RolesAllowed` annotations which are placed on methods definitions to control who has permission to invoke them, as shown in the example presented in Figure 23. In this example only users with either the *Seller* or *Director* roles are allowed to call the method *getCustomer*. Java EE enforces RBAC dynamically at runtime by checking if users indeed play one of the specified roles.

This approach conveys some relevant limitations:

#### Users identification

There is no control neither on the identification of who is invoking protected methods nor on the identification of who is being instantiated. This means that any *Seller* and any *Director* are allowed to get access to any *Customer*.

#### Awareness

The checking process is only dynamically verified at runtime. This means that programmers cannot statically verify if application code in fact respects the enforced RBAC policies.

```

15     @RolesAllowed({"Seller", "Director"})
16     public static Customer getCustomer(int customerId) {
17         Customer c = null;
18         // ...
19         return c;
20     }
21     // .. more code

```

Figure 23. Enforcement of RBAC in Java EE.

**[Fischer, '09]**

In [Fischer, '09], Fisher et al. introduce Object-sensitive RBAC (ORBAC), an extension of RBAC to be used with object-oriented programming languages. The goal is to address limitations of current RBAC model and associated frameworks as the one provided by Java EE. Instead of controlling access at the class level, ORBAC supports access control at the level of individual objects, allowing a finer-grained access control than Java EE. Additionally, ORBAC provides a type system that statically ensures that a program is in accordance with a specified ORBAC policy, preventing programmers from writing application code not aligned with the established policies. ORBAC addresses these limitations by allowing roles and privileged operations to be parameterized by a set of index values which are used to distinguish users of the same role. Figure 24 presents the case of Figure 23 but now based on ORBAC. The *RoleParam* annotation on the *cid* variable (customer Id) indicates that *cid* will be used as an index in role annotations within the class. *Requires* annotation is equivalent to Java EE *@RolesAllowed* annotation but uses additional meta-data to statically allow *Seller<customerId>* or *Director<customerId>* to invoke *getCustomer* only. *@Returns* annotation is similar to a post condition asserting that the returned *Customer* object has a *cid* role parameter variable which is equal in value to the customer identification passed to the method.

```

25     @RoleParam public final int cid; // customer Id.
26     @Requires(roles={"Seller", "Director"}, param={"customerId", "customerId"})
27     @Returns(roleParams="cid", vals="customerId")
28     public static Customer getCustomer(@RoleParam final int customerId) {
29         Customer c = null;
30         // ...
31         return c;
32     }
33     // .. more code

```

Figure 24. Enforcement of RBAC in ORBAC.

**[OASIS, '12]**

XACML [OASIS, '12], as previously described, comprises two main components, PEP and PDP. PEP is responsible for enforcing the decisions of PDP. Basically, every PEP comprises some logic to communicate with PDP and then uses some business logic to accomplish its task whenever authorization is granted. Therefore, whenever a modification in a policy implies a modification on the business logic, there is no other solution than update the business logic in advance.

XACML does not give any guidance about any aspect of business logics. Not about how to keep them updated, not about how to promote the awareness of the implemented mechanisms, etc.

## 2.5 Summary

This chapter is focused on the required background to ease the reading and the understanding of this thesis and also on the state of the art. It is organized in four sub-sections, each one addressing a different issue.

The first section presents some basic concepts such as the most relevant access control policies, architectures of FGACM and dimensions of FGACM. There are several types of policies but RBAC is the most used policy to protect sensitive data of relational database applications. The architecture of FGACM may follow one of three possible approaches: centralized, distributed or mixed. Each approach presents advantages and disadvantages. Independently from the followed architecture, FGACM present four additional dimensions, each one with its implications: granularity, awareability, contextuality and adaptability.

The second section presents current tools that are used to build business tiers. Several tools have been devised to develop business tiers but none of them addresses access control. Two types of tools were emphasized: O/RM tools and CLI. From these tools, CLI were chosen as the underlying middleware to interact with RDBMS. CLI provide powerful features if correctly exploited lead also to powerful implementations of FGACM. Performance and several access modes to data are two of the most important features of CLI.

The third section presents JDBC. JDBC is the selected CLI to be used in the proof of concept of the DACA.

The fourth section describes current approaches addressing FGACP. Several approaches are presented aimed at providing access control to data residing on RDBMS. Some are provided by vendors of RDBMS, others have been provided by the academic community and other has been proposed through a standard emanated from OASIS, XACML. The diversity of needs and the diversity of possible solutions lead to the current situation where system architects are frequently pushed to devise their own and specific security solutions. From the presented background and state of the art, there is the evidence that current approaches to deal with access control are based on: tools provided by vendors of RDBMS, query rewriting techniques, extensions to the SQL standard, new programming languages, language extensions and XACML approach. None of the approaches address the dynamic adaptation of FGACM deployed at the client-side applications. As previously mentioned, the adaptation of mechanisms is an avoidable activity to be performed in advance when mechanisms evolve. Moreover, the current research approaches deal mostly with native CRUD expressions only (do not take advantage of other access modes such the ones provided by CLI) and some of them do not support other types of CRUD expressions but Select expressions. Additionally, the freedom provided by current approaches to use any CRUD expression opens the possibility of leaking security gaps.

The next chapter describes the path that has been followed from the CLI until the DACA.



## 3 From Call Level Interfaces Towards the DACA

This chapter explains the research that has been conducted to design the DACA from CLI. Basically three steps were taken. In the first step, the architecture of CLI has been redesigned to define a model able to incorporate schemas of database objects to tackle the impedance mismatch between the relational and the object-oriented paradigms [Pereira, '10b; Pereira, '11b]. In the second step, the model has been adapted to promote the development of reusable business tier components. Finally, in the third step, the outcome of the previous work was used to link access control on business tier components. The main outcome of this third step is the Dynamic Access Control Architecture. The DACA has been devised to implement dynamic FGACM on business tiers based on CLI. Some concepts are common to the three steps and, therefore, they will be presented beforehand to avoid unnecessary repetitions of text and descriptions.

The chapter is organized as follows. Section 3.1 introduces some fundamental concepts for the DACA. Section 3.2 presents the model used to integrate CLI and schemas of relational databases. Section 3.3 presents the model used for building reusable business tier components form CLI. Section 3.4 briefly describes the approach that has been followed to enforce dynamic access control policies at the level of business tiers relying on CLI and, finally, section 3.5 summarizes the present chapter.

### 3.1 Concepts

CRUD expressions and LMS are two key entities of CLI. Both are the entities used to access databases and, therefore, the entities on which FGACM may rely on. To this end, we introduce three concepts to formalize the execution of CRUD expressions: CRUD Schema to formalize CRUD expressions, Business Schema to formalize the necessary services to manage the access to data for the two access modes (Direct Access Mode and Indirect Access Mode) and, finally Business Entity to formalize the software artifact responsible for implementing a Business Schema to execute CRUD expressions. These concepts were devised and developed during the two last steps [Pereira, '12d; Pereira, '11a; Pereira, '11c; Pereira, '12c; Pereira, '12b; Pereira, '13a; Pereira, '13b; Pereira, '13d; Pereira, '13e].

#### 3.1.1 CRUD Schema

There are four types of CRUD expressions (Select, Update, Insert and Delete) each one with its own characteristics. Some characteristics are shared among two or more types but others are not shared. These observations led to question whether it would be possible to formalize an abstract representation for CRUD expressions. To start the process, we present the main characteristics of

CRUD expressions:

#### Types of CRUD expressions

There are four types of CRUD expressions conveying different properties. Analyzing their properties, we see that they may be organized in two major groups. One group, known as *Reading*, comprises the Select expression type only and the other group, known as *Updating*, comprises the remaining three types of CRUD expressions. This organization is mainly derived from the fact that Select expressions return relations and the other CRUD expressions do not. Thus, there is a clear difference on the services to be provided for each group.

#### Runtime values

Beyond their types and the syntax of the SQL language, applications use other entities during the building process of CRUD expressions. These entities are a sort of variables whose values are set at runtime and are used by applications to exchange data with RDBMS. There are three types of variables: attribute list, column list and clause list.

#### Attribute set

The attribute set is characteristic only of CRUD expressions of type Select. The attribute set represents returned values by Select expressions. Attribute set is commonly known as the attribute list. Attribute sets are not optional on CRUD expressions of type Select. Every Select expression has one attribute set.

#### Column set

The column set is characteristic of CRUD expressions of type Insert and Update. They are used to dynamically define runtime values for column lists. Column lists contain the values to be inserted or updated on database columns. Columns sets are not mandatory.

#### Clause set

The clause set is characteristic of CRUD expressions of type Select, Update and Delete. They are used to dynamically set runtime values of clause conditions. Clause sets are not mandatory.

#### Result

Unlike the Select expression, the remaining three types of CRUD expressions modify the state of databases. Delete expressions delete rows, Update expressions update rows and Insert expressions insert new rows. After being executed and the database state modified, client applications are informed about the number of modified rows.

From these characteristics, a simplified formalization for CRUD expressions is now presented. There are two types of CRUD expressions. The type Reading is formalized by the attribute set and the clause conditions set. The type Updating is formalized by the column set, the clause condition set and the number of affected rows, as follows

*Type = {Reading, Updating}*

*Reading = {Att, CC}*

*Updating = {Col, CC, Result}*

Figure 25 shows three CRUD expressions representing the two types of CRUD expressions and representing different combinations of runtime values. The first CRUD expression is of type Reading and the correspondent CRUD Schema comprises all attributes of table *Categories*, one runtime value for clause conditions. The second CRUD expression is of type Updating and the correspondent CRUD Schema comprises three runtime values for the attribute set. The third CRUD expression is of type Updating and the correspondent CRUD Schema comprises one runtime value for the attribute set and one runtime value for the *where* clause condition. The relevancy of CRUD Schema concept is not restricted on being a formalization method of CRUD expressions. Another relevant aspect derives from the fact that the relationship between CRUD Schemas and CRUD expressions is 1 to many. An indeterminate number of CRUD expressions may share the same CRUD Schema. Figure 26 shows an example of two CRUD expressions: both are Select expressions, both share the same attribute set and both have no values defined at runtime. CRUD expressions sharing the same CRUD Schema are herein known as sibling CRUD expressions.

```

Select *
  From Categories c
  Where c.CategoryID=?;

Insert into Categories
  Values (?, ?, ?);

Update Categories
  Set categoryName=?
  Where CategoryID=?;

```

Figure 25. Three CRUD expressions with different combinations of CRUD Schemas.

```

--CRUD expression 1
Select *
  From Categories;

-- CRUD expression 2
Select *
  From Categories c
  Where c.CategoryId=10;

```

Figure 26. Two sibling CRUD expressions.

The presented concept of CRUD schema confines the scope of CRUD expressions to sibling CRUD expressions only. This restriction is acceptable and adequate when a tight binding between services to be provided and CRUD expressions is a requirement. In situations in which this tight binding is not a key requirement, the CRUD Schema concept is too restrictive preventing the grouping of similar CRUD expressions that are not siblings. To overcome this situation, the concept of CRUD Schema is extended to support not one but one or more CRUD Schemas. This approach may be used in any situation whenever there is the need to optimize the number of CRUD Schemas. Jayapandian and Jagadish [Jayapandian, '08] have concluded that a large number of CRUD expressions “*can potentially be composed from a given set of related schema elements*”. Beyond not

being CRUD Schemas, Schema elements [Yu, '06] are neither concerned about access control. Anyway, their approach may be used to optimize the number of CRUD Schemas. Returning to the main point, depending on the needs and requirements, there are two approaches to formalize CRUD Schemas, which are herein known as closed approach and open approach.

### **3.1.1.1.1 Closed approach**

In the closed approach, CRUD Schemas are only used to formalize sibling CRUD expressions. The closed CRUD Schema approach has the advantage of conveying a complete schema awareness of each CRUD expression. As a disadvantage, each CRUD Schema is not flexible to accommodate CRUD expressions with different CRUD Schemas. If a CRUD expression is formalized through a different CRUD Schema a new CRUD Schema is needed.

### **3.1.1.2 Open approach**

Unlike the closed approach, the open approach is designed for managing several CRUD Schemas. CRUD Schemas supported by the same open CRUD Schema are herein known as sibling CRUD Schemas. Sibling CRUD Schemas are characterized by sharing their types of CRUD expressions and their attributes sets. Only runtime parameters may vary from CRUD Schema to CRUD Schema. This means that the variations between CRUD Schemas are limited to the parameters whose values are defined at runtime: column set and clause set. The open approach has more flexibility than the closed approach, this way leading to advantages during the development process of business tiers and also after their deployment (at runtime). Next follows a description for each advantage.

#### Development

The flexibility of the open approach increases the opportunity to reuse existent CRUD schemas when a new CRUD expression is needed. Therefore, the open approach minimizes the number of the needed CRUD schemas to support a set of CRUD expressions.

#### Runtime

If the architecture of business tiers supports the deployment of CRUD expressions at runtime, then the open approach will minimize the maintenance activities at the business tiers level.

In spite of these significant advantages, the open approach also conveys some drawbacks. Among them, two are emphasized:

#### Awareness

The flexibility of the open approach is obtained by providing services able to support any number and any type of runtime parameters. This flexibility requires programmers to master schemas of runtime parameters. This need is not necessary if the closed approach is used because the schemas for the runtime parameters are tailored to one CRUD schema only.

#### Security

The flexibility of the open approach conveys more freedom to use more CRUD expressions.

Whenever security is considered a key aspect, a greater supervision is needed to know which CRUD expressions are being used.

Listing 5 shows four CRUD expressions none of which is sibling. When using the closed approach, four CRUD Schemas are required: one for each CRUD expression. When using the open approach, two CRUD Schemas are required: one for the two first CRUD expressions and another for the two remaining CRUD expressions.

```

-- CRUD expression 1
Select *
  from Categories;

-- CRUD expression 2
Select *
  from Categories
  where CategoryId=?;

-- CRUD expression 3
Insert into Categories
  values (?, ?);

-- CRUD expression 4
Insert into Categories
  values (?, ?, ?, ?);

```

Listing 5. Four examples of CRUD expressions.

Summarizing, a CRUD Schema comprises five independent parts:

- a mandatory type schema - the CRUD type (Reading or Updating);
- the attribute set (only for Select expressions);
- an optional clause set (open or closed approach) – for setting the runtime values for the conditions used inside SQL clauses, such as the “*where*” and “*having*” clauses;
- an optional column set (open or closed approach) – for setting the runtime values for the column list of Insert and Update CRUD expressions;
- a mandatory result schema for Insert, Update and Delete CRUD expressions – to retrieve the number of affected rows whenever CRUD expressions are executed.

### 3.1.2 Business Schema

Business Schema (BS) leverages the CRUD Schema concept to formalize the set of necessary services to be provided to manage the execution of CRUD expressions organized by CRUD Schemas. It comprises several services among which are emphasized: 1) access to data through the direct and indirect access modes and 2) services to manage the scrolling process on LMS. These services are customized to address specific requirements needs. For example, when dealing with access control, Business Schemas are driven by access control and, therefore, have to be arranged in order to be in accordance with the established FGACP.

When dealing with access control, the Direct Access Mode is concerned with the authorized CRUD expressions written in the native SQL language and the Indirect Access Mode is concerned about the actions on LMS. Table 3 shows a possible definition for the permissions on an LMS derived from the CRUD expression *Select a, b, c, d from table*. This access matrix [Lampson, '74] like

representation defines for each attribute of this LMS, which actions (read, update, insert, delete) are authorized. *delete* action is authorized in a tuple basis and, therefore, it is executed as an atomic action on all attributes. In situations where access control is not provided, and when LMS are updatable, all actions on LMS are available to be used by programmers of application tiers. This example is access control oriented but the concept of Business Schema is not tied with any specific purpose. The only purpose of Business Schema is to provide a formalization process to reorganize the services provided by CLI to access data residing on LMS.

	a	b	c	d
<b>Read</b>	yes	no	yes	yes
<b>Update</b>	no	yes	no	yes
<b>Insert</b>	yes	yes	no	no
<b>delete</b>	yes			

Table 3. Example of a table of permissions in a LMS (Indirect Access Mode).

### 3.1.3 Business Entity

Business Entities (BE) are software artifacts (classes) responsible for managing the execution of

```

19 // read
20 public int rA() throws SQLException {
21     return rs.getInt("a");
22 }
23 public int rC() throws SQLException {
24     return rs.getInt("c");
25 }
26 public int rD() throws SQLException {
27     return rs.getInt("d");
28 }
29
30 // update
31 public void uB(int value) throws SQLException {
32     rs.updateInt("b", value);
33 }
34 public void uD(int value) throws SQLException {
35     rs.updateInt("d", value);
36 }
37
38 // insert
39 public void iA(int value) throws SQLException {
40     rs.updateInt("a", value);
41 }
42 public void iB(int value) throws SQLException {
43     rs.updateInt("b", value);
44 }
45
46 // delete
47 public void delete() throws SQLException {
48     rs.deleteRow();
49 }

```

Figure 27. Partial example of how to implement the permissions of Table 3 on LMS.

CRUD expressions through the implementation of Business Schemas. Therefore, Business Entity is a general concept to be used for the building process of software classes. As Business Schemas, Business Entities are not oriented to address any specific requirement, such as access control. In each particular context, Business Entities, as Business Schemas, are customized to address the required needs. Instances of Business Entities are herein referred to as Business Workers (BW). Figure 27 partially presents an example based on JDBC to show a Business Entity that wraps a ResultSet (rs - LMS) and enforces the permissions defined in Table 3. Only attributes *a*, *c* and *d* are readable (line 20-28). Only attributes *b* and *d* are updatable (line 31-36). Only attributes *a* and *b* are insertable (line 39-44). Rows are deletable (line 48).

## 3.2 Modelization of Call Level Interfaces

In spite of their individual successes, the object-oriented and the relational paradigms are simply too different to bridge seamlessly, leading to difficulties informally known as *impedance mismatch* [David, '90]. The diverse foundations of the object-oriented and the relational paradigms are a major hindrance for their integration, being an open challenge for more than 45 years [Cook, '05]. The challenge derives from the multiplicity of aspects that need to be bridged across both paradigms: imperative languages versus declarative languages; compilation and execution performance versus search performance; classes, algorithms and data structures versus relations and indexes; transactions versus *threads*; null pointers versus null for the absence of value [Cook, '05], and finally, inheritance versus specialization. The impedance mismatch thus presents several challenges for developers of database applications, where often both paradigms are found. These challenges are especially noticeable in environments where production code is under strict development deadlines, and where (timely) code development efficiency is a major concern. In order to cope with the impedance mismatch issue, several solutions have emerged, among them CLI are herein emphasized. In spite of their relevancy, CLI present several drawbacks as previously described. The modelization of CLI was the first step to overcome some of the drawbacks and it was addressed in the following papers [Pereira, '10a; Pereira, '10b; Pereira, '11b; Pereira, '06; Óscar Narciso Mortágua Pereira, '05a; Óscar Narciso Mortágua Pereira, '05b].

### 3.2.1 Motivation

This section aims to emphasize common drawbacks regarding the utilization of CLI. The modelization process is not concerned with access control but mainly with the integration process of the relational and the object-oriented paradigms. In this context, the main drawbacks of CLI are organized in four categories [Pereira, '10b; Pereira, '11b]:

- 1- The process for editing CRUD expressions;
- 2- The process for reading data from returned relations;
- 3- The process of updating databases through updatable LMS;
- 4- Protocols of LMS regarding their usability.

One again, JDBC is used as a representative of CLI. Figure 28 presents a simple example, which comprises some of the drawbacks related to categories 1), 2) and 3). This example is used in the following paragraphs to describe JDBC drawbacks:

### Linkage

There is no easy way to link CRUD expressions and their results to the application they assist. JDBC provides services to ease the integration of object-oriented applications and relational databases but relevant issues are not overcome such as string concatenation (Figure 28: line 22-24) and the conversion between relational and object-oriented paradigms (Figure 28: lines 27, 28, 30).

### Edit

The editing process of CRUD expressions and access to their results is tricky and error-prone.

CRUD expressions are constructed by concatenating strings and access to their results is achieved by reading attribute by attribute in a row by row basis. Some of the most usual errors are:

#### Concatenation errors

Whenever CRUD expressions are built from concatenated strings there are several types of errors that are easily made. The most common and very often very difficult to detect are missing spaces between lines (Figure 28: lines 22, 23) and missing spaces between substrings as the missing before “and” (Figure 28: line 23).

#### Type mismatch

Programmers need to master CRUD Schemas to be able to use the correct data type when accessing attributes of LMS. Any type mismatch error is only detected at runtime leading to an increased effort to deploy business tiers error free.

#### Misspelled attribute name

Programmers need to master CRUD Schemas to be able to use the correct attribute name when accessing attributes of LMS. Any type misspelled name is only detected at runtime leading to an increased effort to deploy business tiers error free.

### Debug

Previous errors cannot be checked for correctness at compile time, addressed in [Gary, '07]. None of the previous errors can be caught at compile time demanding great accuracy while editing the source code to prevent additional time on testing, debugging and future maintenance.

### Maintenance

CRUD expressions are awkward regarding their maintenance, addressed in [Andy, '08]. CRUD expressions (building process and execution) depend on many different entities grouped in three classes: SQL syntax, services of CLI and database schemas. While SQL syntax and services of CLI can be considered stable, database schemas are dynamic entities. Database schemas change for many reasons. Some of the most common reasons are:

- An initial error on conceptual model or logical model;
- The emerging of new requirements, which usually happens several times during the development process and even also after the deployment process of database applications.



Any simple change in a database schema may involve a huge work on updating not only the strings that encode the affected CRUD expressions but also the schema of the returned relations and, therefore, the name of attributes that are used in the indirect access mode.

### SQL injection attacks

CRUD expressions are vulnerable to SQL injection attacks, addressed in [Gregory, '05]. This issue is not addressed in this thesis.

### LMS usability

LMS have dozens of states, dealing with different combinations of LMS instantiations, directions, accesses, updates, etc. The developer is before a huge task to become aware of how to use LMS. LMS comprise several distinct protocols not organized in distinct interfaces, conveying the idea that everything is possible in anytime. For example, ResultSet interface is composed by more than 200 methods and 10 attributes. Each ResultSet state has its own usage protocol gathering a subgroup of all methods of the ResultSet interface. While Read and Delete protocols do not comprise a start and an end instruction, Update and Insert protocols always have a start instruction (implicitly for Update and explicitly for Insert) and an end instruction. Besides the starting and the ending instructions, the main issue for Update and Insert protocols is that the cursor cannot be moved from the current selected row while the protocol is being executed. If the cursor is moved from the selected row while the protocol is being executed, the protocol will be aborted and previous changes are discarded from the in-memory of LMS. In order to overcome some of these difficulties we will present an approach where each protocol is executed through a dedicated interface this way improving ResultSet usability.

```

21 void product(int categoryId, float unitsInStock) throws SQLException {
22     sql="select * from Products" +
23         "where categoryId=" + categoryId + "and "+
24         "unitInStock<" + unitsInStock +";";
25     rs=st.executeQuery(sql);
26     while (rs.next()) {
27         productId=rs.getInt("productId");
28         productName=rs.getString("produtName");
29         // other attributes
30         rs.updateInt("unitsOnOrder",unitsOnOrder(productId));
31         // update other attributes
32         rs.updateRow();
33         // more code
34     }
35 }

```

Figure 28. Typical JDBC/CLI drawbacks.

Some of the aforementioned drawbacks have already been individually addressed by other authors as previously cited. The modelization of CLI proposal in this work constitutes an integrated and unified alternative to overcome all the aforementioned drawbacks, except for the SQL injection attack.

### 3.2.2 Proposed Approach for the Modelization of CLI

The modelization process does not cover all functionalities of CLI but only those directly related with the execution of CRUD expressions such as those related to the access modes of CLI: the Direct Access Mode and the Indirect Access Mode. The modelization process aims at tackling the aforementioned drawbacks of CLI. The approach is based on a model and on a tool from which Business Entities are automatically built. Figure 29 presents the model to represent CLI, herein referred to as the CRUD-Model. This model clearly identifies the main sub-functionalities of CLI and aggregates them in independent interfaces: IExecute, ILMS, IResult and ISet. Next follows a description for each interface:

#### IExecute

IExecute interface comprises services to execute CRUD expressions using the Direct Access Mode. Beyond the execution of CRUD expressions, this interface is responsible for setting the runtime values of clause conditions for all types of CRUD expressions.

#### ILMS

ILMS interface is used to access to functionalities of LMS and it is available only when CRUD expressions are of type Reading. One of its main functionality is the management of the Indirect Access Mode. ILMS comprises several interfaces:

#### IReadability

IReadability interface comprises one interface, IRead, to read data from LMS. This interface is used for read-only and updatable LMS. Methods of IRead are driven by the schema of the returned relation and, as such, are semantically oriented and type-safe.

#### IUpdatability

IUpdatability interface comprises several interfaces to manage updatable LMS:

- IDelete: comprises all methods associated with the delete protocol;
- IInsert: comprises all methods that are needed to control the insert protocol;
- IUpdate: comprises all methods that are needed to control the update protocol;
- IWrite: comprises all the methods associated with the write protocol. These methods are driven by the schema of the returned relation and, as such, are semantically oriented and type-safe;
- IRead: comprises all methods associated with the read protocol. These methods are driven by the schema of the returned relation and, as such, are semantically oriented and type-safe.

#### IScrollability

IScrollability interface comprises two interfaces to manage the two possibilities for scrolling policies:

- IScrollable: comprises all methods associated with scrollable LMS. The methods are only present if the LMS is scrollable;
- IForwardOnly: comprises all methods associated with forward-only LMS. The methods are only present if the LMS is forward-only.

ISet

ISet interface is used to set the runtime values for the column set of Insert and Update expressions.

IResult

IResult interface is used to retrieve the number of affected rows when a CRUD expression of type Updating is executed.

Figure 30 presents a block diagram for the modelization process of CLI. Basically, the architectural CRUD-Model accepts as input CRUD expressions and some additional metadata to build Business Entities responsible for managing CRUD expressions. This implementation is very similar to the one presented in [Pereira, '10b; Pereira, '11b]. From CRUD expressions and from complementary metadata (for example, Scrollability policy and the Updatability policy to be used), the architectural model is responsible for validating the correctness of CRUD expressions, for inferring the CRUD Schemas and also for building automatically the source code for Business Entities in accordance with the CRUD-Model.

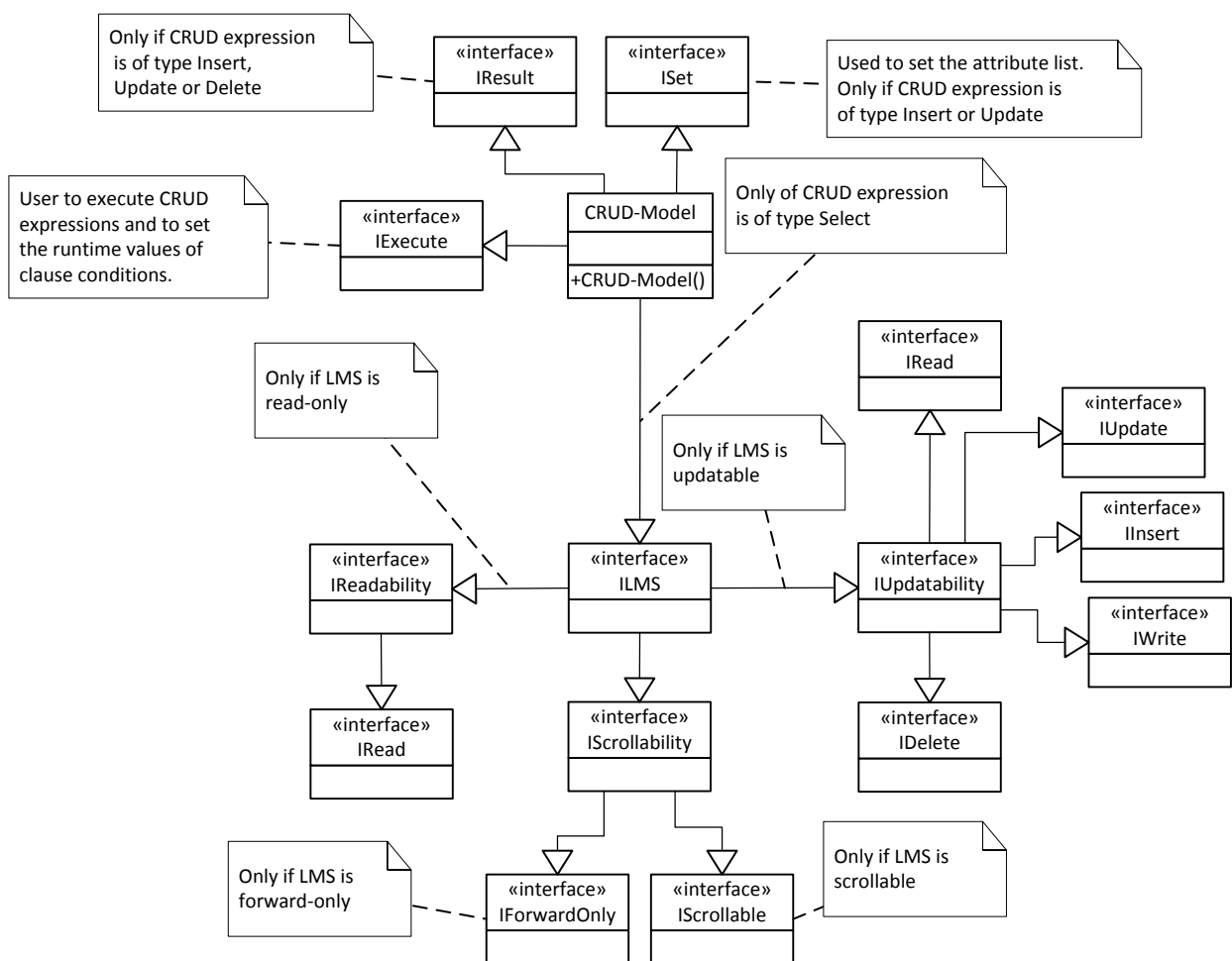


Figure 29. Business Schema for the modelization of CLI: CRUD-Model.

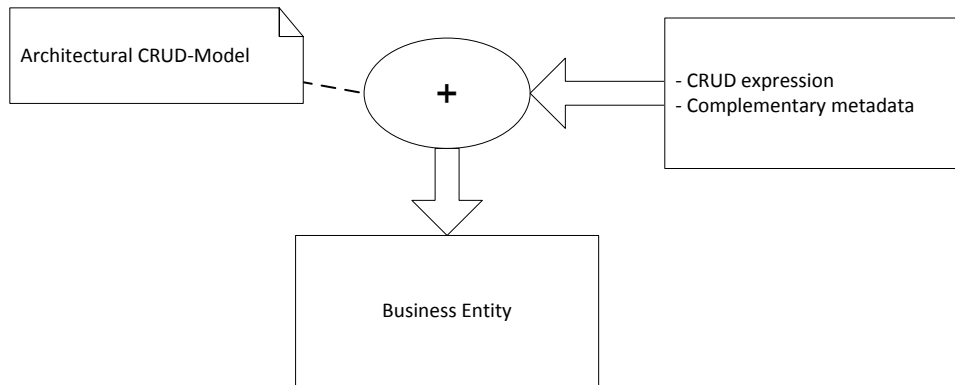


Figure 30. Block diagram for the modelization process of CLI.

```

8 public class Product implements IExecute, IScrollability, IUdatability {
9     private String sql; private ResultSet rs;
10    private Statement st; private Connection conn;
11
12    + public Product() {...}
13
14
15    ④ @Override
16    - public void execute(int categoryId, int unitsInStock) throws SQLException {
17        sql="select * from Products where categoryId=" + categoryId + " and "+
18        "unitsInStock<" + unitsInStock +";";
19        rs=st.executeQuery(sql);
20    }
21
22    ④ @Override
23    - public boolean moveNext() throws SQLException {
24        return rs.next();
25    }
26
27    ④ @Override
28    - public int productId() throws SQLException {
29        return rs.getInt("productId");
30    }
31
32    ④ @Override
33    - public String productName() throws SQLException {
34        return rs.getString("productName");
35    }
36    // .. read remaining attributes
37    ④ @Override
38    - public void unitsOnOrder(int unitsOnOrder) throws SQLException {
39        rs.updateInt("unitsOnOrder",unitsOnOrder);
40    }
41    // update remaining attributes
42    ④ @Override
43    - public void updateRow() throws SQLException {
44        rs.updateRow();
45    }
46    ④ @Override
47    - public void cancelUpdate() throws SQLException {
48        rs.cancelRowUpdates();
49    }
50    // other protocols

```

Figure 31. Partial view of a Business Entity based on the CRUD-Model.

Next follows an example to show a real case to implement a Business Entity. Figure 31 presents a partial view of a Business Entity aimed at managing the CRUD expression

```
Select * from Products
Where categoryId=? and unitsInStock<?
```

The CRUD expression is compiled-on-the-fly (complementary metadata) and LMS is forward-only and updatable (complementary metadata). To accomplish these requirements, the presented Business Entity implements the following interfaces: IExecute, IScrollability (IForwardOnly) and IUpdatability (IRead, IUpdate, IInsert and IDelete).

Now that a Business Entity has been presented, we show, from the application tier point of view, the use of that Business Entity (see Figure 32). The drawbacks presented in this section for CLI are clearly overcome by CRUD-Model. The following aspects are emphasized:

CRUD expressions

CRUD expressions are now automatically encoded inside strings after being validated by the CRUD-Model (Figure 31: line 19-20). Previous errors associated with CRUD expressions are no longer a concern.

Source code

There is no need to write any source code. From CRUD expressions and from selected metadata, source code of Business Entities is automatically built in accordance with the CRUD-Model (Figure 31).

```

22 public void product() throws SQLException {
23     Product p = new Product ();
24     p.execute(categoryId, unitsInStock);
25     IRead r=p;
26     IUpdate u=p;
27     IWrite w=p;
28     while (p.moveToNext()) {
29         productId=r.productId();
30         productName=r.productName();
31         // other attributes
32         w.unitsOnOrder(unitsOnOrder(productId));
33         // update other attributes
34         u.
35     }
36 }
37 }
38 }
39 }
40 }

```

Figure 32. Example shown in Figure 28 but based on the CRUD-Model.

### LMS usability

Functionalities of LMS are now organized around interfaces (Figure 32: lines 25-27) and the access methods are semantically driven and type safe (Figure 32: line 29-32). From the open pop-up window on line 34 we see that interface *IUpdate* provides two methods: *updateRow* and *cancelUpdate*.

To achieve these results, there is the need to devise a tool similar to the one used in [Pereira, '10b; Pereira, '11b]. With this tool, programmers need only to write CRUD expressions and define some additional metadata to overcome all the presented drawbacks of CLI. Then, the tool automatically builds Business Entities to manage the execution of CRUD expressions. Each Business Entity manages its own CRUD expression.

## **3.3 Componentization of CLI**

The componentization process of CLI is mainly concerned with the building process of reusable and adaptable business tier components. Componentization of CLI was addressed in the following papers [Pereira, '11a; Pereira, '11c; Pereira, '12b; Pereira, '13a; Pereira, '13b; Pereira, '13e]. Good programming practices advise the development of database applications relying on a multi-tier architecture. The three tier architecture is the most widespread one comprising the application tier, the database tier and the middle tier known as the business tier. The business tier may provide a clear separation (technological, administrative and organizational) between host databases and client applications. Database applications of some complexity may comprise hundreds of CRUD expressions to deal with business requirements. Very often they cannot be inferred from any data model that may eventually be available (database schema). This leads to situations where the development and maintenance processes of business tiers are very tedious and exhaustive. Programmers are pushed to write similar source code for each CRUD expression, mainly for Select expressions with a long attribute list. There should exist a methodology to relieve programmers from these tedious, exhaustive and error-prone processes. To address these gaps, a research has been conducted to devise reusable business tier components based on CLI.

### **3.3.1 Components**

Component-based development is a key topic in software engineering [Bachmann, '00; Heineman, '01; Szyperky, '02]. Component-based development aims to compose software artifacts from other pre-built software artifacts [Heineman, '01]. At the end, a final system is not built as a unique block but as a composite of software artifacts known as components [Kung-Kiu, '07]. A key aspect for the success of any component is its capability of being reused and adapted [Bracciali, '05]. In reality, despite the relevancy of the postulates, reutilization and adaptation of components raise several technological difficulties and, maybe not least important, easily gathers voices against their adoption. For example, component replacement has some disadvantages conveying an impact on the overall system. Some of the disadvantages are [Costa, '07]:

#### Loss state lost

When a component is replaced, its state may be lost. To avoid this situation, the new

component must be initialized in the state of the replaced component.

#### System availability

During the replacement process, the component or even system availability may be affected. To avoid component unavailability, components need to be decoupled from client components, eventually by using proxies.

#### Performance decay

Performance decay usually occurs during the replacement process. Components being replaced need to be deactivated and substituent components need to be activated and initialized. Performance decay seems to be an unavoidable consequence of the replacement process. An effort is necessary to minimize the negative interference of the replacement process of components.

In order to avoid component and system unavailability, several approaches may be followed to dynamically adapt them at run-time, which is one of the crucial aspects of Component Based Software Engineering (CBSE) [Bracciali, '05]. The adaptation of components should comprise not only the configuration process but mainly the replacement of old services and also the definition of new services in a seamlessly way. Another key issue is the reuse of computation [Elizondo, '10], which maximizes the reuse of computation to address different computational needs. Among the several proposed approaches, models@run.time [Blair, '09] is emphasized. Models@run.time are playing an increased role in software systems of organizations from which critical decisions are taken, such as airports, power plants and hospitals. These systems have to be available 24 hours a day and 7 days a week and are expected to safely adapt to varying runtime contexts. Software models@run.time give the answer to this requirement. In [Blair, '09] says: *“Runtime adaptation mechanisms that leverages software models extend the applicability of model-driven engineering techniques to the runtime environment.”* In Model Driven Engineering models are used to formalize and render complex systems in a manageable way for humans and for computers. Software models@run.time keep these important features and step forward by incorporating the specification of the systems they formalize, Bran Selic in [Blair, '09]. Through the specification and through the runtime context software models@run.time support dynamic adaptation. Software models@run.time may be seen as an important contribution to the field of autonomic computing [Kephart, '03].

### **3.3.2 Adaptation Process**

The adopted adaptation process uses the same model as the one presented for the modelization process, CRUD-Model, but with a slight difference. Now CRUD expressions are not statically compiled on Business Entities but are dynamically deployed and passed to them through their constructors. This will be explained during the next paragraphs. Thus, the similarity between the two class diagrams eliminates the need to present a new class diagram. Another difference exists in the process used to automatically build source code for Business Entities. While in the modelization process, the source code for each Business Entity was built from one CRUD expression and from some additional metadata, in the componentization process, the source code for Business Entities is

automatically built from metadata only. CRUD expressions are dynamically deployed in a later stage, at runtime.

The differences between the two processes have been described, and now the focus is on the componentization process of CLI. The adaptation process of business tier components is basically focused on the capability to support new CRUD expressions. To achieve this goal there are basically three dimensions to be addressed, which are herein referred to as the Service Allocation, Service Composition and Service Scope. Service Allocation is mainly concerned with deploying CRUD expressions at runtime to address new business needs. Service Composition is mainly concerned about creating Business Entities to address new business needs. Service Scope is mainly concerned about the extent of the services to be provided by each Business Entity.

### **3.3.2.1 Service Allocation**

Service Allocation proposes the deployment process of CRUD expressions to be accomplished at runtime. Unlike the approach used for the modelization process, where CRUD expressions are statically allocated to Business Entities at compile time, the Service Allocation allows the deployment process of CRUD expressions to be accomplished at runtime this way introducing a new dimension in the adaptation process: Service Allocation promotes the deployment of CRUD expressions based on policies. Policies may be used to deploy CRUD expressions driven by countless possibilities such as users profiles, driven by security policies and driven by the runtime context.

### **3.3.2.2 Service Composition**

Service Composition is mainly concerned on the building process of Business Entities. Services of Business Entities are formalized by Business Schemas, which are mainly based on the CRUD-Model presented for the modelization process. Service Composition may be accomplished following two different approaches: static approach [Pereira, '11a; Pereira, '11c; Pereira, '13a; Pereira, '13e] and the dynamic approach [Pereira, '12b].

#### Static Service Composition

When using the Static Service Composition, Business Entities are statically built before the deployment process of business tier components. Business tier components built from the Static Service Composition address a business area, such as accountability or sales. Then, at runtime, CRUD expressions are deployed following any established policy. Figure 33 presents a block diagram for the Static Service Composition. In a) business tier components are statically built from Business Schemas and in accordance with an architectural model based on the CRUD-Model. In b), after being deployed, the component accepts CRUD expressions in accordance with any established policy. To be effective, components relying on the Static Service Composition must provide a variety of Business Entities able to manage all the needed CRUD expressions in order to minimize or even prevent future maintenance activities.

#### Dynamic Service Composition

When using the Dynamic Service Composition, Business Entities are dynamically built at runtime to address any runtime business needs. Similarly to the Static Service Composition,



CRUD expressions are also deployed at runtime following any established policy. Figure 34 presents a block diagram for the Dynamic Service Composition. Business Engine is the entity responsible for building Business Entities dynamically at runtime from Business Schemas deployed by a Monitoring Framework or any other entity skilled to achieve the same result.

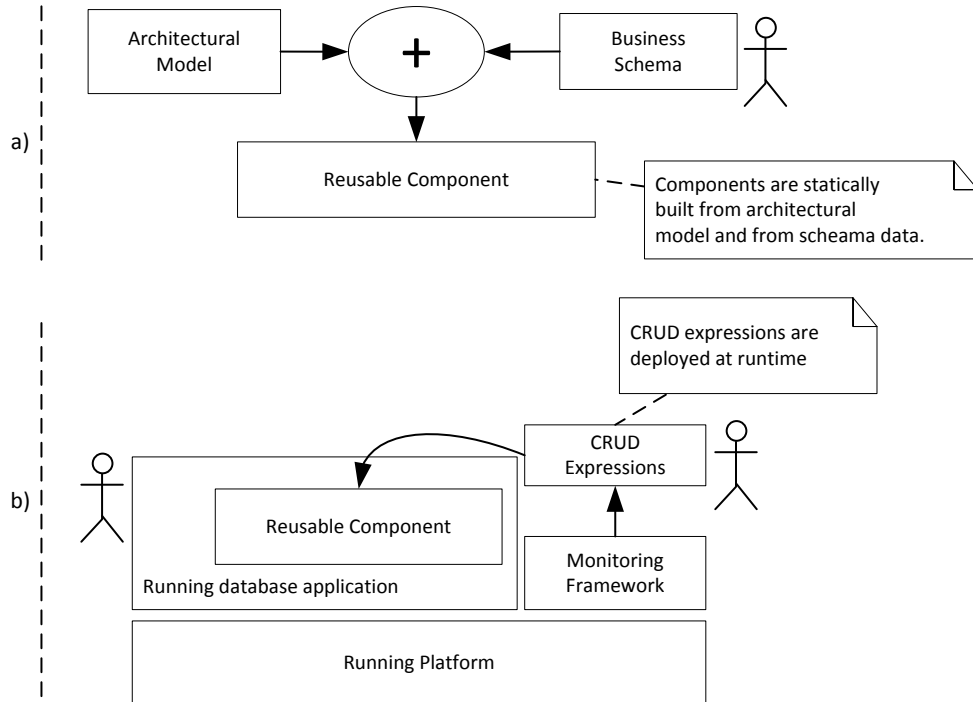


Figure 33. Block diagram for the static approach: a) service composition and b) service allocation.

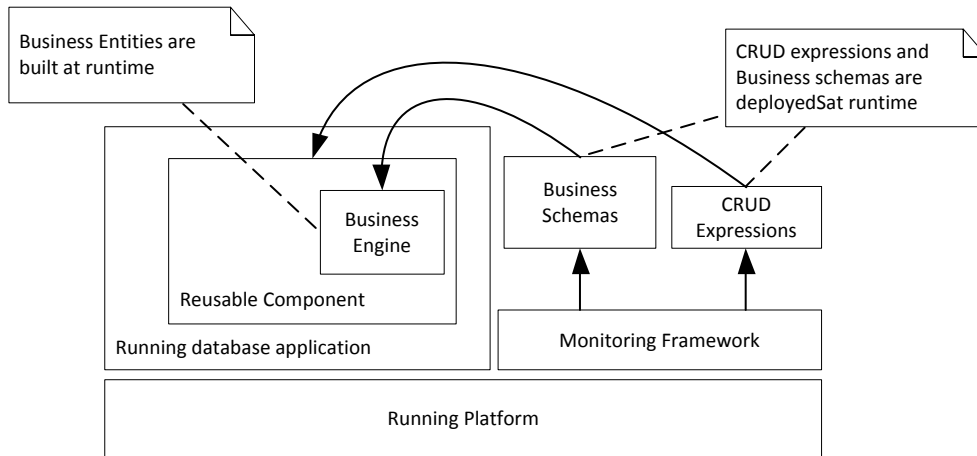


Figure 34. Block diagram for the dynamic service composition.

### 3.3.2.3 Service Scope

Service scope concept is based on the CRUD Schema concept to promote two different approaches for the scope of Business Schemas: Unique Business Schema (based on the open approach of CRUD Schemas) and Multiple Business Schemas (based on the closed approach of CRUD Schemas). The Unique Business Schema [Pereira, '11c; Pereira, '13e] is used whenever there is the need to minimize the number of CRUD Schemas and the Multiple Business Schema [Pereira, '11a; Pereira, '13a] is used when there is the need to keep CRUD Schemas closely aligned with CRUD expressions.

#### Unique Business Schema

Business tier components based on the Unique Business Schema approach provide a unique and fixed set of Business Entities responsible for managing all the necessary CRUD expressions. The Unique Business Schema approach is specially effective when CRUD Schemas are only known at runtime and the Dynamic Service Composition is not recommended. To address these constraints, the Static Service Composition process needs to build three unique Business Entities each one wide enough, based on the open CRUD Schema approach, to support any foreseen CRUD expression. The three Business Entities to be made available are one for all Select expressions, one for all Update and Insert expressions and, finally, one for all Delete expressions.

#### Select expressions

Each business tier component has its own Business Entity for managing all Select expressions. The Business Entity is built to address one or more business areas, such as accountability or sales. IRead and IWrite must comprise all the needed attributes to support the addressed business area. Thus, the attributes are not proprietary of any Select expression but in reality they are shared by all CRUD expressions. Each CRUD expression makes use of the attributes formalized by its CRUD schema. Figure 35 schematically shows a set of CRUD expressions, each one requesting a subset of the attributes that are made available through the IRead and IWrite interfaces. Additionally, to support any number and any type of runtime parameters, the open approach is used for the CRUD schema. Thus, the method to set the runtime values for clause conditions (IExecute interface) must have as argument *object[]* of type *Object* to support values of any data type and in any quantity.

#### Insert and Update expressions

Unlike the previous Business Entity, the Business Entity responsible for managing all insert and update expressions is shared by all business tier components. The CRUD Schema follows the open approach and is characterized by two main methods. One method to set the runtime values for clause sets (IExecute interface) having as argument *object[]* of type *Object* to support values of any data type and in any quantity. And another method to set the runtime values for column sets of Insert and Update expressions (ISet interface) having *object[]* as argument to support values of any data type and in any quantity.

#### Delete expressions

Similarly to the previous Business Entity, the Business Entity responsible for managing all delete expressions is shared by all business tier components. The CRUD Schema follows the open approach and is characterized by one main method. To set the runtime values for

clause conditions (IExecute interface). Similarly to the previous methods, it also has *object[]* of type *Object* as argument to support values of any data type and in any quantity.

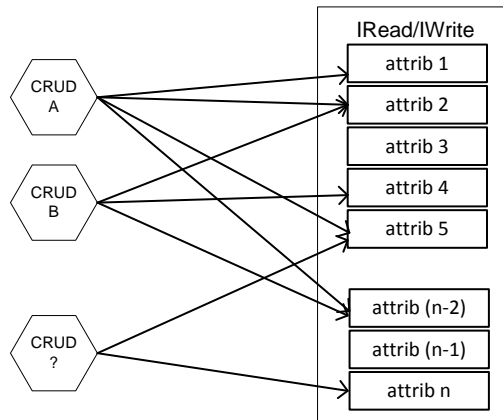


Figure 35. Attributes shared by all CRUD expressions.

**Multiple Business Schemas**

The Multiple Business Schema approach is specially effective when CRUD Schemas are known at development time for the Static Service Composition, or at runtime for the Dynamic Service Composition. There will be as many Business Schemas as necessary. Each Business Schema generates one Business Entity able to manage any CRUD expression whose schema is contained by the implemented Business Schema. Figure 36 shows an example similar to the one shown in Figure 35 but following the multiple business schema. Here there are several CRUD Schemas for Select expressions where each CRUD Schema owns its particular Business Entity. Some CRUD

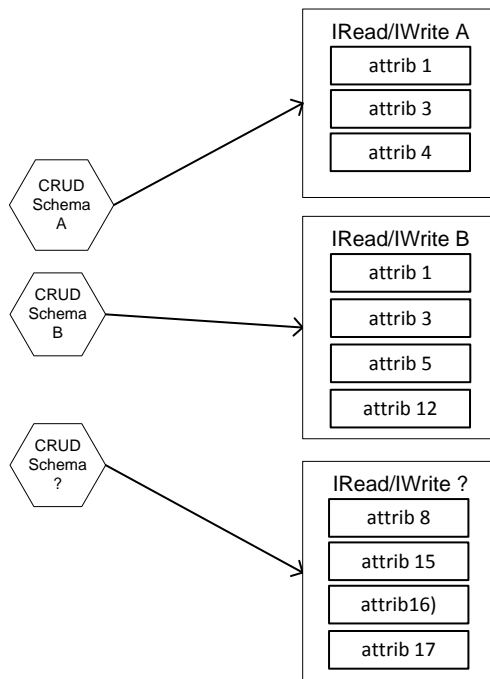


Figure 36. Example of one Multiple Business Schema implementation.

Schemas may share some attributes, as CRUD Schema A does with CRUD Schema B, but each one has its own IRead and IWrite interfaces. The Multiple Business Schema approach may also be used to differentiate other properties of CRUD Schemas, such as runtime values for clause conditions and runtime values for column lists avoiding this way the need to use *Object[]* as argument.

### 3.3.2.4 Business Schema

Business Entities are built from Business Schemas only. As such, Business Schemas need some additional attention for the componentization process of CLI. Business Schemas comprise several interfaces as shown in Figure 29. In spite of their complexity, Business Schemas are very easy to be defined because most of the interfaces are written only once or comprise one method only. For example, IScroll, IForwardOnly, IResult, IInsert, IUpdate and IDelete are unique and shared by all Business Schemas. Regarding IExecute and ISet, each one contains one method only. The only interfaces entailing some complexity are IRead and IWrite. The effort for their definition is required during the Service Composition phase. These interfaces comprise the getter and setter methods for the attributes of LMS. But the effort is actually significantly less than it might seemed, because the methods belonging to each IWrite interface are automatically inferred by the Business Engine from the correspondent IRead interface. For example, if an IRead interface comprises methods *Integer a()* and *String b()* then the correspondent IWrite interface comprises the methods *void a(Integer value)* and *void b(String value)*. Thus, Business Engine relieves programmers from the need to write the IWrite interface.

## 3.4 Access Control

Modelization and componentization overcome important drawbacks of CLI. Nevertheless, they are not enough to address access control let alone the implementation of evolving FGACM. FGACM need a fine tune control on the access to data residing on RDBMS. CLI provide two distinct modes to access data: the Direct Access Mode and the Indirect Access Mode. Both access modes need to be governed by FGACM. Access control on the Direct Access Mode is about controlling the authorized CRUD expressions. Access control on the Indirect Access Mode is about controlling the authorized actions at the cell level (row – column) of LMS. Access control was mainly addressed in [Pereira, '12d; Pereira, '12c; Pereira, '13d]. Basically, the access modes of CLI were wrapped by services driven by FGACM. These issues are thoroughly described in the next chapter.

## 3.5 Summary

The evolution from CLI concept till the DACA was presented in a three step approach. Initially, the fundamental concepts were introduced: CRUD Schema, Business Schema and Business Entity. These concepts are used since the very start till the final definition of the DACA. They define the basic entities from which drawbacks of CLI are overcome. Regarding the three step approach towards the DACA, during the first step, a model has been defined to bridge the gap between the object-oriented and the relational paradigm. During the second step, an architecture has been defined for the building process of reusable business tier components. These components are built combining three concepts: Service Allocation, Service Composition and Service Scope. These concepts, when combined with each other, open several possibilities for the development and

adaptation processes of business tier components. Another relevant aspect is that IRead and IWrite interfaces are the only interfaces requiring some effort during the Service Composition phase. They comprise the getter and setter methods for the attributes of LMS. But the effort is actually minimized because the methods belonging to each IWrite interface are automatically inferred by the Business Engine from the correspondent IRead interface. A brief introduction was done to the access control approach on CLI. Now, the DACA will be main topic of the next chapter.



## 4 DACA: Dynamic Access Control Architecture

In this chapter a new architecture, herein known as the DACA, is proposed for building business tiers, based on Call Level Interface, embedded with FGACM and driven by dynamic adaptation. The DACA is only focused on FGACM and it does not address policies or even models. The DACA is an architecture for the implementation of dynamic FGACM, which is completely decoupled from policies and models. We first introduce an overview of the approach to be followed to implement FGACM, then follows the general architecture and finally details are given for each main component of the DACA. The DACA leverages all previous researches conducted around CLI, models and components, to provide a solution relying on CLI to enforce evolving FGACM on business tier components. The DACA also leverages and deeply relies on other previous researches [Pereira, '12d; Pereira, '12c; Pereira, '13d].

This chapter is organized as follows. Section 4.1 introduces the approach followed to implement FGACM at the level of business tiers. Section 4.2 presents the general architecture of the DACA. Section 4.3 presents the main components of the DACA and, finally, section 4.4 summarizes this chapter.

### 4.1 Fine-grained Access Control Mechanisms

The DACA relies on CLI and, as such, FGACM are implemented at the level of CLI on business tier components. Hence, the implementation of FGACM on business tier components based on CLI cannot be disconnected from the services provided by CLI to access data residing in RDBMS. As previously presented and described, CLI provide several modes to interact with data residing on RDBMS. Among them, two were emphasized and hereafter recalled:

- Direct Access Mode – through this mode, CLI provide services to allow CRUD expressions to be encoded inside strings using the native SQL language or eventually the RDBMS SQL language;
- Indirect Access Mode – through this mode, CLI provide services to allow the execution of any of the provided protocols at the level of LMS of CLI: read, update, insert and delete protocol.

These two access modes are the key points from which FGACM are defined and implemented. FGACM use Business Schemas as the key entities to control the access to data. Business Schemas wrap and exploit the access modes of CLI to expose a set of access modes driven by FGACM. Therefore, the concept of Business Schema is redefined to address requirements of FGACM.

## 4.2 General Architecture

FGACM in the DACA are implemented at the client-side level and specifically at the level of business tiers based on CLI. The implementation process of FGACM need to cope with one main research question and three second level research questions previously announced: dynamicity of FGACM, security, awareness of FGACM and preservation of CLI advantages.

### Dynamicity

FGACM need to be dynamically adapted at runtime to address evolving FGACP. This requires that the client-side systems have the ability to be locally adapted in accordance with the established FGACP. Moreover, as the FGACM are deployed in each client-system, there is no other way but provide a central system from which the directive for the FGACM to be implemented on the client-side systems are issued.

### Security

Current tools allow users to write any CRUD expression. Due to the endless expressiveness of the SQL language this freedom may lead to security violations. Thus, the DACA needs to ensure that all issued CRUD expressions are in accordance with the established FGACP.

### Awareness

FGACM need to be implemented in a way to convey a complete awareness about the established FGACM during the development process of application tiers. This awareness relieves programmers from mastering the established FGACP and the correspondent FGACM.

### Preservation of CLI advantages

To keep CLI advantages, the DACA needs to ensure two aspects. The first one is that the services of CLI must be kept and provided by the DACA. The second one is that performance of CLI must also be kept. To cope with these requirements services of the DACA need to be closely aligned with the services of CLI and, additionally, they must induce a minimum processing overhead.

These requirements will be all addressed in this chapter.

### 4.2.1 Phases of the DACA

The DACA needs to cope with several requirements, among which the awareness of FGACM at development time of application tiers and the dynamic implementation of FGACM at runtime are emphasized. To address these requirements, the DACA operation is split at least in two phases: one responsible for the static representation of FGACM and the other one for the dynamic adaptation process of FGACM. The first phase takes place while application tiers are being developed, while the second phase takes places at runtime. From these two phases a third phase is inferred, which is concurrent and independent from the other two, during which metadata of FGACM are defined and updated.

The general architecture of the DACA is presented as a block diagram in Figure 37. Lines connecting components with small circles on their edges represent socket connections and the ending arrows identify the components playing the server role. The general architecture may be



organized and presented using several distinct perspectives. The perspective presented in Figure 37 is based on the three main phases just presented:

Configuration phase

The configuration phase of the DACA is responsible for keeping metadata of FGACM updated in accordance with the established FGACP. The metadata of FGACM are the source from which FGACM are automatically built and kept updated.

Extraction phase

During the extraction phase, the DACA creates data structures, which are used to convey to programmers of application tiers a complete awareness of the established FGACM. These data structures need to be statically represented while programmers write source code to access data residing on RDBMS to prevent them from writing source code not aligned with the established FGACM.

Running phase

During the running phase, metadata of FGACM is used to build the correspondent FGACM dynamically. Any modification in the metadata leads to an automatic updating process on the implemented FGACM. Moreover, CRUD expressions are also deployed at runtime in accordance with the established FGACP. This deployment process is important because it will be used to relieve programmers from writing CRUD expressions and, therefore, prevent any security violation.

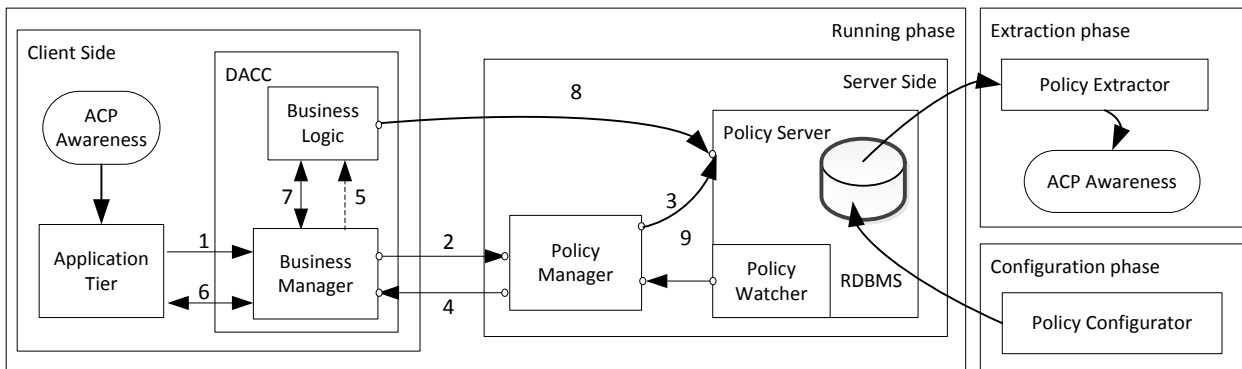


Figure 37. General architecture of the DACA.

**4.2.1.1 Configuration Phase**

The configuration phase is focused on the configuration and maintenance processes of metadata of FGACM, which are stored in a server, in our case in a RDBMS. This RDBMS may or may not be the same where the protected data reside. The configuration process may occur at any time even when database applications are running, after their deployment. The only constraint is that the definition of FGACM needed during the development and maintenance phases of application tiers, have to be defined before they are requested and therefore, before the occurrence of the

Extraction Phase. Then, during the Runtime Phase metadata of FGACM may evolve to address new security needs.

The configuration process is carried out by using a component herein referred to as the Policy Configurator. Policy Configurator is used to define and keep metadata of FGACM updated, at any time, independently from the other two phases. The metadata is stored in the Policy Server and has its origin on the used policy model (not addressed by this thesis) and on the granted permissions organized as Business Schemas and the associated CRUD expressions. Figure 38 represents the permission concept in the DACA. Permission is the authorization to use a Business Schema and a set of CRUD expressions to be managed by that Business Schema.

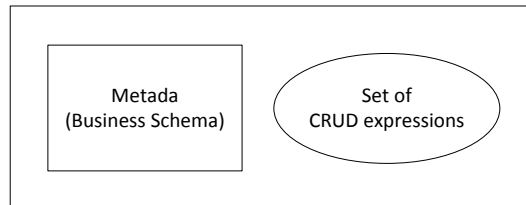


Figure 38. Concept of permission in the DACA.

#### 4.2.1.2 Extraction Phase

The extraction phase is focused on formalizing FGACM as programming data structures so that they can be statically represented by IDE and then used during the development process of application tiers. These data structures restrict application tier programmers to only use authorized accesses to RDBMS. This way, programmers become aware of FGACM at development time of application tiers and not at compilation time or at runtime. Basically, the data structures comprise roles (if a RBAC policy is used) and the associated permissions. To successfully accomplish this phase, metadata of FGACM need to be previously defined in the configuration phase.

The extraction phase is carried out by using a component herein referred to as the Policy Extractor. Policy Extractor is used only during the development process and also on the maintenance process of application tiers.

#### 4.2.1.3 Running Phase

The running phase is focused on adapting the client-side FGACM in accordance with the established FGACP. Any modification in the FGACP during the running phase needs to be translated into metadata of FGACM to be then automatically enforced in the client-side components.

During the running phase, database applications are running. There are two main blocks: a client side block and a server side block, see Figure 37.

##### Client side block

The client side block comprises a unique component herein known as the Dynamic Access Control Component (DACC). The DACC is responsible for providing application tiers with all the services they need to access data residing in RDBMS and based on the two following principles:

1) the provided services to access to data are driven by FGACM and 2) the provided services are closely aligned with the standard services of CLI.

#### Server side block

The server side block comprises two main components, herein known as the Policy Manager and the Policy Server. The server side block is mainly focused on managing the metadata of FGACM and also on making them available to the client side block.

### **4.2.2 General Operation of the DACA**

The DACA comprises three main components: the DACC, the Policy Server and the Policy Manager, see Figure 37. The DACC is responsible for the adaptation process of business tiers to implement FGACM, the Policy Manager is a broker between the Policy Server and the DACC and the Policy Server stores metadata of FGACM and keeps the Policy Manager informed (through the Policy Watcher) about any modification in the metadata of FGACM. The DACA general operation is as follows:

- The Policy Server and the Policy Watcher are started. The Policy Server and the Policy Watcher play server roles and are responsible for managing metadata of the FGACM to be enforced.
- The Policy Manager is started. It establishes a connection with the Policy Server (Figure 37: 3) and registers itself in the Policy Server. This way, the Policy Server becomes aware of all running instances of Policy Managers. This is important because in case the Policy Server goes down and after restarting up, there is the need to know the running instances of Policy Managers and how to connect to them (see two next points).
- The Policy Manager closes the connection and waits for a connection to be established by the Policy Watcher.
- The Policy Watcher establishes a connection with the Policy Manager (Figure 37: 9).
- Application tiers create instances of the DACC (Figure 37: 1) and authentication is provided: username, password and application identification.
- The DACC establishes a connection with the Policy Manager (Figure 37: 2) to become registered and closes the connection.
- The DACC waits for a connection from Policy Manager.
- The Policy Manager registers the DACC in the Policy Server (Figure 37: 3).
- The Policy Manager establishes a connection with the Business Manager (Figure 37: 4).
- The Policy Manager identifies and selects the metadata of FGACM (Figure 37-3) to be implemented by the DACC and send them to the DACC (Figure 37: 4).
- The DACC automatically builds a Business Logic (Figure 37: 5).
- Application tiers ask the DACC to manage the execution of CRUD expressions on their behalf (Figure 37: 6).
- The Business Manager contacts the Business Logic (Figure 37: 7) to manage application tiers requests. The Business Logic sends CRUD expressions to a RDBMS (Figure 37: 8), which may be shared or not with the Policy Server, and returns to application tiers the results of their execution (Figure 37: 7,6).
- Any modification in the established metadata of FGACM is internally managed by the Policy Server. The Policy Watcher sends them to the Policy Manager (Figure 37: 9) which then

sends them to the Business Manager (Figure 37: 5) which, finally, adapts the Business Logic to new FGACM (Figure 37: 5).

### 4.3 The DACA Components

In this section a more detailed explanation is given for each constituent component of the DACA.

#### 4.3.1 The DACC

To keep advantages of CLI, the DACC relies on and is closely aligned with CLI. It is responsible for building and maintaining business tiers driven by evolving FGACP. In reality, the DACC are realizations of business tiers and are the only components of the DACA that application tiers use to access data residing in RDBMS. As such, the DACC architecture was designed to address two main requirements:

- The DACC provide an environment to developers of application tiers as similar as possible to those provided by CLI;
- The DACC are dynamically and continuously adapted at runtime to be kept aligned with evolving FGACP.

These requirements led to an architecture of the DACC based on two entities loosely coupled: the Business Manager and the Business Logic. While Business Manager ensures the implementation of all services shared by all DACC (it is a static component as all the remaining components of DACA, except Business Logic), the Business Logic is dynamically, at runtime, adapted to build business tiers driven by FGACP. Basically, the Business Logic comprises a set of Business Entities built at runtime and on a set of authorized CRUD expressions to be used on Business Entities. Figure 39 presents a simplified block diagram of the DACC. The main characteristics to be emphasized are:

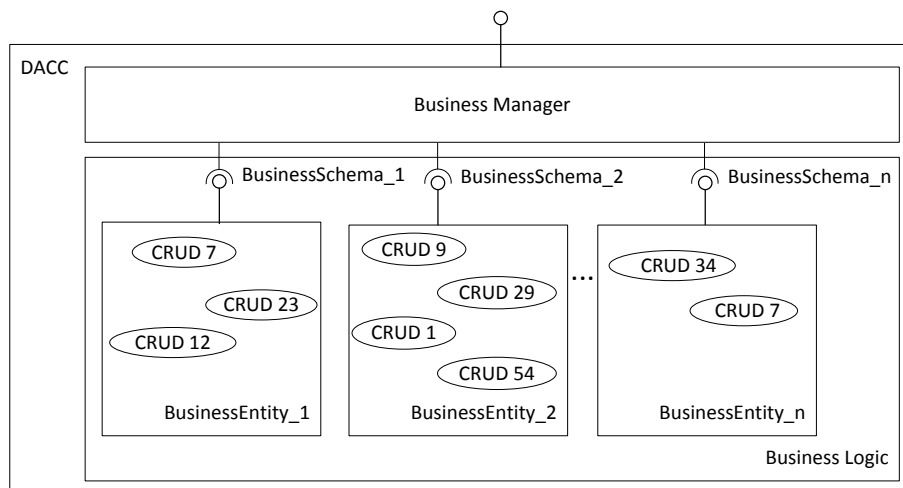


Figure 39. Simplified block diagram of DACC.

#### Access to services

The DACC provides an interface through which applications tiers access its services. The

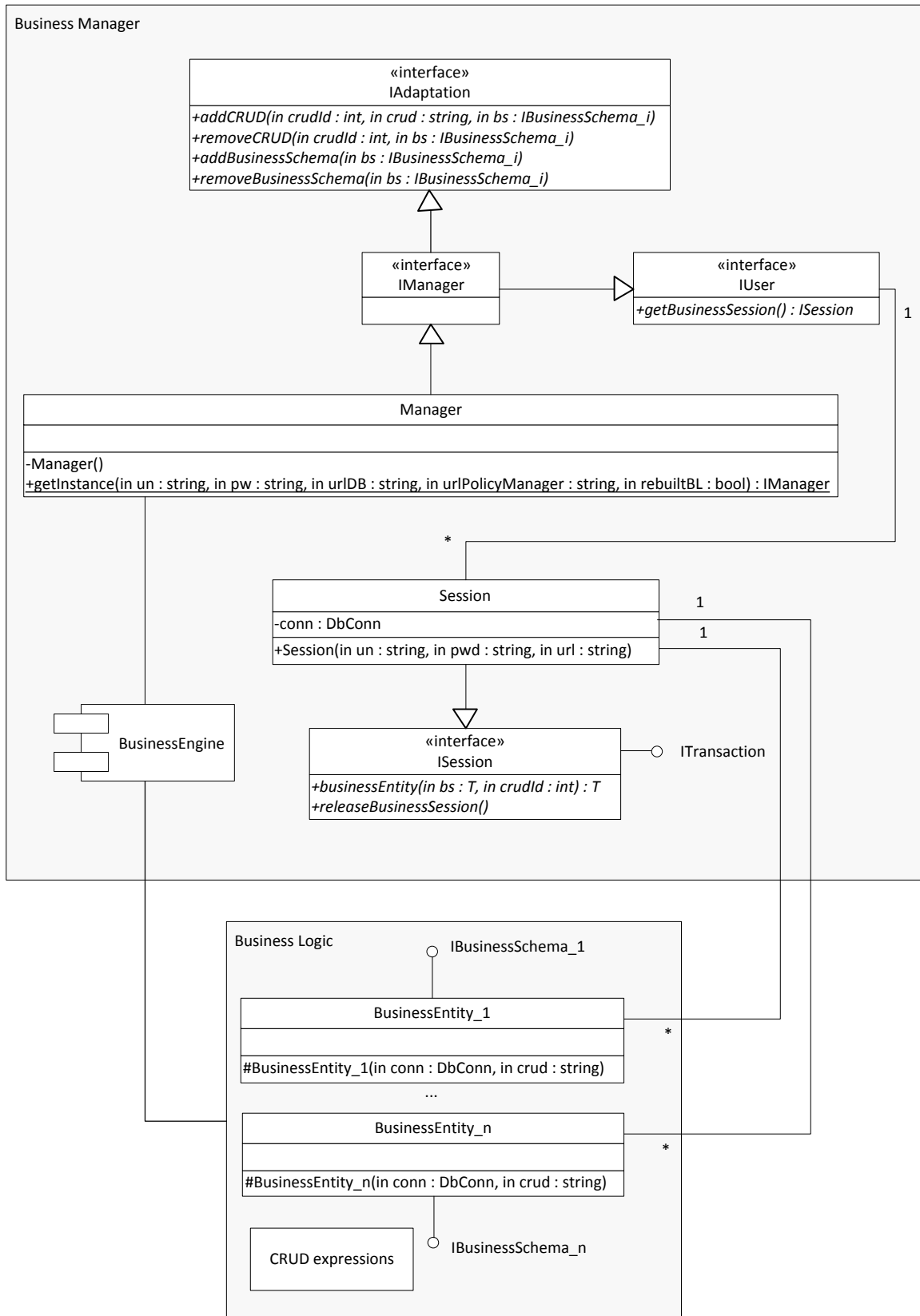


Figure 40. Class diagram of DACC.

services include the ones related to the access modes and also the ones related to complementary services such as instantiation of Business Entities.

### Access to Business Logic

Business Logic is not directly accessed from the DACC outside. The access to entities contained in the Business Logic, such as Business Entities, is managed by an entity herein known as Manager.

### Business Logic

The Business Logic is a container where Business Entities and CRUD expressions are kept and maintained in the client-side of the DACA. Business Entities and CRUD expressions are dynamically inserted and removed from Business Logic in accordance with the established permissions and, therefore, in accordance with the FGACP. In association with each Business Entity there is a set of CRUD expressions that are made available to be executed using the Direct Access Mode of CLI.

Figure 40 presents the class diagram of the DACC. The Business Manager is the top component and the Business Logic is the bottom component. These components, as we will show, are loosely coupled to allow a seamless dynamic adaption process of Business Logic at runtime without raising any runtime exception. Hereafter, each component is described.

## **4.3.1.1 Business Manager**

The Business Manager, see Figure 40, is a component responsible for providing several services organized in two main types of functionalities:

- a functionality to implement the adaptation process of Business Logic to implement the FGACM;
- a functionality to allow application tiers to order the execution of CRUD expressions on their behalf.

Next follows a thoroughly description for each entity of the Business Manager.

### Manager

From application tiers perspective, the Manager is the entry point of the DACC. The DACC is instantiated through the *getInstance* method. This method has as arguments the user authentication, the *url* to the host RDBMS (Figure 37: 8), the *url* to the Policy Manager (Figure 37: 2) and a condition to evaluate if Business Logic is to be rebuilt. This last argument is important mainly during the development process of application tiers to avoid unnecessary rebuilding processes of the Business Logic. After the initiation of the instantiation process, the sequence to be followed is the one described in 4.2.2. Manager implements IManager interface, which implements the IAdaptation and the IUser interfaces

### IAdaptation

The IAdaptation interface provides services for the adaptation process of DACC or, in other words, the interface provides services to keep Business Logic aligned with the established metadata of FGACM. This interface is implemented as a socket (Figure 37: 4) to allow the adaptation process to be carried out by system processes running in different space memories even in other computers as shown in Figure 37. *addCrud* and *removeCRUD* are used to grant and

deny permissions to execute CRUD expressions on Business Entities. *addBusinessSchema* and *removeBusinessSchema* are used to build and remove typed objects (Business Entities implementing Business Schemas) responsible for enabling the execution of CRUD expressions on Business Schemas.

### IUser

The IUser interface provides a single method, *getBusinessSession*, to create a new session. A session is mainly characterized by owning a private database connection represented by the Connection interface described in 2.3.3. Then, through sessions, Business Entities may be instantiated and CRUD expressions are executed.

### ISession

The ISession interface provides two methods – *businessEntity* and *releaseBusinessSession*.

#### businessEntity

The *businessEntity* is a generic method used to create new instances of any Business Entity. It is defined as a generic method to promote two important aspects: first, with the implemented approach, only one method is needed to instantiate any Business Entity; second, the instantiation process is type-safe. These two aspects are important because Business Entities are not defined at compile time. *businessEntity* accepts a Business Schema and a CRUD expression identification as arguments and returns an instance of a Business Entity that implements the Business Schema provided as argument. Basically, the *businessEntity* operation follows the sequence next described:

- Business Logic is searched to find if there is a Business Entity implementing the requested Business Schema;
- If it there is not, an exception is raised. It means that the user is not authorized to use the requested Business Schema;
- Otherwise;
  - It is checked if the user has permission to use the requested CRUD expression on that Business Entity;
  - If it has no permission, an exception is raised;
  - Otherwise:
    - The Business Entity (class) is loaded into memory;
    - Through reflection, an instance of Business Entity is created;
    - An instance is returned to the application tier.

This strategy clearly implements a loosely coupled dependency between Business Manager and Business Logic which is an essential issue to allow the dynamic adaptation of Business Logic. This approach was used for the first time in [Pereira, '12b] and then reused in [Pereira, '13b; Pereira, '13d]. Beyond the dynamic adaptation, this loosely coupled dependency also allows the development process of application tiers to be independent from the implementation of Business Entities. All programmers need are the data structures built from the metadata of FGACM extracted during the Extraction phase.

#### releaseBusinessSession

The second method, *releaseBusinessSession*, is used to release a session being used. There is

no guide to what to do with the connection object. Anyway, the establishment of connections with RDBMS is widely known as being non-negligent regarding the time consuming and the CPU consuming. As such, it is recommended to use a pool of connections to avoid the overhead induced by the waste of resources when connections are activated and deactivated. In case of not being possible to develop a manager for the pool of connections, there are some API providing this type of service, such as [Oracle, '12d; Waldman, '12].

#### ITransaction

ITransaction interface provides all the required services to manage database transactions. The interface is defined at the Session level because transactions are managed at the connection level. This means that each connection, at any time, may only have one active transaction.

#### BusinessEngine

The Business Engine is another key component in the DACC. The Business Engine is responsible for managing the contents of Business Logic: Business Entities and CRUD expressions. For example, regarding the Business Entities, Business Engine automatically creates the source code for them from Business Schemas, compiles the source code and stores them inside the Business Logic. The Business Engine is the entity responsible for keeping Business Logic updated and in accordance with the established metadata for the FGACM

### **4.3.1.2 Business Logic**

The Business Logic is mainly composed by two types of entities: CRUD expressions to be made available to application tiers and Business Entities to be made available to application tiers to manage the execution of CRUD expressions on their behalf. CRUD expressions and Business Entities are dynamically inserted and removed from the Business Logic at runtime to address evolving FGACP. The dynamic adaptation and the implementation of FGACM are two fundamental dimensions of the DACA each one appealing to different needs. While CRUD expressions are basically Strings, Business Entities are classes and, therefore, are more complex entities. Next follows a more detailed description of the Business Logic implementation.

#### **4.3.1.2.1 General Approach for the Business Logic**

In chapter 2 a description is given for CLI and also for the functionalities of JDBC. JDBC class diagrams are presented in Figure 18, Figure 19, Figure 20, Figure 21 and Figure 22. Architecture of JDBC, and CLI in general, clearly does not promote the development of business tiers driven by dynamic FGACP. Next follows a detailed explanation for the approach that has been used on the Business Logic sub-component to implement dynamic FGACM

#### Dynamic Adaptation

When the focus is the implementation of FGACM, dynamic adaptation is about the continuous updating process of FGACM in accordance with the established metadata of FGACM for each user. This means that the Business Logic is dynamically built from scratch and thereafter continuously updated in accordance with the defined FGACM for each user. One of the possibilities to address this requirement is the use of complementary components addressing



models@run.time. Components based on models@run.time have the ability to be continuously adapted at runtime to address evolving needs.

#### Access modes of CLI

CLI are not tailored to address any kind of access control. They have a fixed set of services allowing programmers to freely access data in RDBMS. To tackle this gap, the only possibility to implement FGACM on CLI is by wrapping and adapting the two access modes of CLI in accordance with the FGACP.

Thus, the dynamic adaptation of the access modes provided by CLI is the key aspect to be addressed to promote the implementation of dynamic FGACM.

### **4.3.1.2.2 Architecture**

Architecture of the Business Logic is now described. It is represented by a model and it and some services are configurable to address FGACM. Figure 41 shows the class diagram for Business Entities driven by FGACM and Figure 42 shows the class diagram for LMS also driven by FGACM. These diagrams are clearly derived from the CRUD-Model previously presented. Some adaptations were enforced to allow the implementation of FGACM.

#### BusinessEntity

Business Entity is the fundamental software artifact of the Business Logic. Business Entities are programming classes responsible for the execution of CRUD expressions. They are formalized through a model represented in Figure 41 and are the entities dynamically built at runtime to implement FGACM. A Business Entity accepts at instantiation time a connection to the host database (*DbConn*) and the CRUD expression to be executed. Each Business Entity implements one Business Schema. Business Schemas are represented by programming interfaces and are herein referred to as *IBusinessSchema* interface.

#### IBusinessSchema

*IBusinessSchema* characterizes the services to be provided by Business Entities. There are three facets:

- one for Select expressions: comprises *IExecute* and *ILMS* interfaces;
- one for Insert and Update expressions: comprises *IExecute*, *ISet* (optional) and *IResult* interfaces;
- one for Delete expressions: comprises *IExecute* and *IResult* interfaces.

#### IExecute

*IExecute* has two facets:

- one for the closed CRUD Schema approach (only one *execute* method is implemented except the last one)
- another for the open CRUD Schema approach (any number of overloaded *execute* methods).

The *execute* methods are responsible for the execution of CRUD expressions and therefore to control the use of the Direct Access Mode of CLI. The arguments are used for setting the runtime

values for clause conditions of all types of CRUD expressions. The last method cannot be used in the closed CRUD Schema approach because it behaves as an unbounded overloaded method. A method with the signature *execute(in params[]: object)* allows the caller to pass any number of parameters and of any data type, this way being in disagreement with the closed CRUD Schema concept.

**IResult**

The IResult interface provides a method for retrieving the number of affected rows as a direct consequence of an Insert, Update or Delete expression execution through the Direct Access Mode.

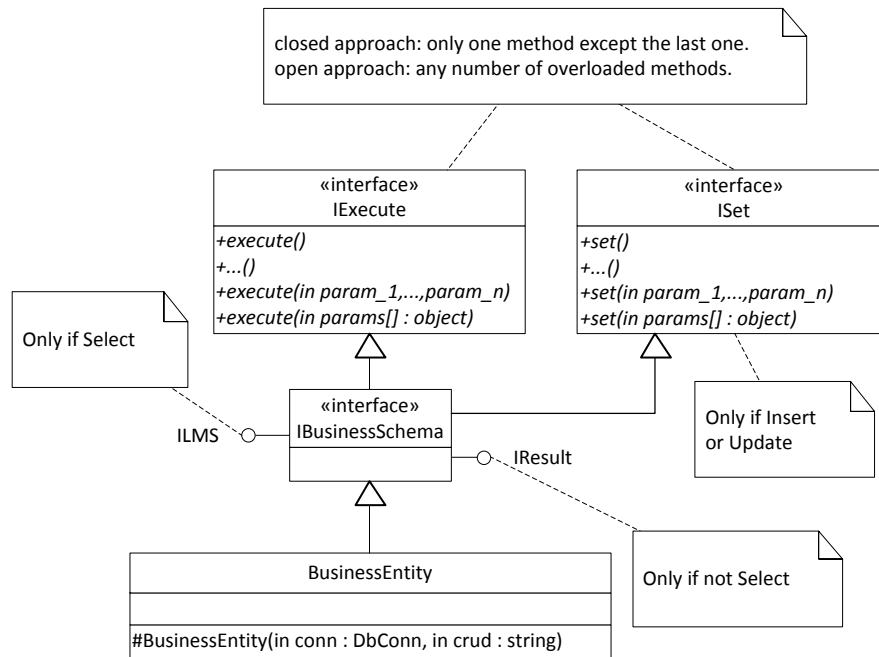


Figure 41. Business Entity class diagram.

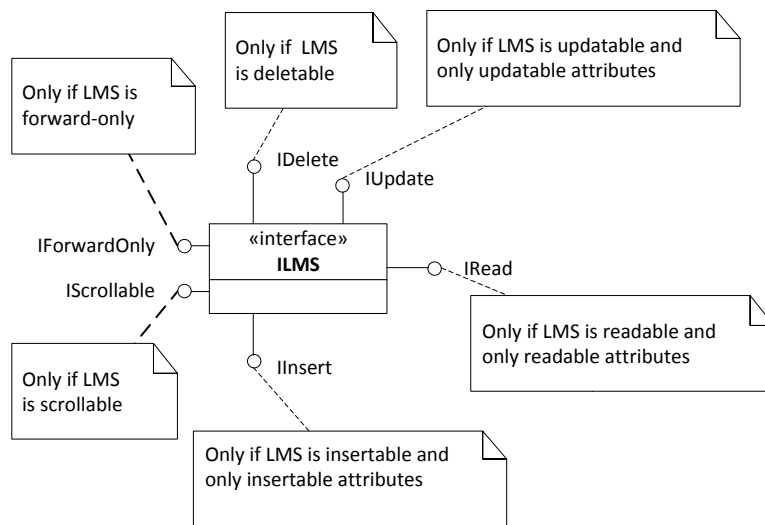


Figure 42. ILMS class diagram for LMS.

### ISet

ISet has two facets:

- one for the closed approach (only one *set* method is implemented except the last one);
- one for the open approach (any number of overloaded *set* methods).

The *set* methods are used to set the runtime time values for column sets of Insert and Update expressions.

### ILMS

ILMS interface is used to implement FGACM on the Indirect Access Mode of CLI and also to

define the scrolling policy, see Figure 42. Unlike the approach followed in the modelization process and in the componentization process of CLI, each action (read, update, insert and delete) is individually configured in accordance with the established access control policies, as the example shown in Table 3. ILMS comprises 6 sub-interfaces, IRead, IUpdate, IInsert, IDelete, IForwardOnly and IScrollable. Next follows a description for each sub-interface.

#### IRead

The IRead interface provides methods to only read the authorized attributes. The attributes that are not authorized to be read cannot belong to the IRead interface.

#### IUpdate

The IUpdate interface is present only if the update protocol is authorized. IUpdate interface provides methods to only update the authorized attributes. The attributes that are not authorized to be updated cannot belong to the IUpdate interface. Additionally, IUpdate interface comprises the required methods for managing the update protocol – start and commit updates.

#### IInsert

The IInsert interface is present only if the insert protocol is authorized. IInsert interface provides methods to only insert the authorized attributes. The attributes that are not authorized to be inserted cannot belong to the IInsert interface. Additionally, it comprises the required methods for managing the insert protocol – start and commit insertions.

#### IDelete

The IDelete interface is present only if the delete protocol is authorized. IDelete interface provides methods to delete rows of LMS.

#### IForwardOnly

The IForwardOnly interface comprises all the methods associated with forward-only LMS.

#### IScrollable

The IScrollable interface comprises all the methods associated with scrollable LMS.

This presentation of ILMS and its sub-interfaces clearly shows that the implemented FGACM is clearly defined at the attribute level and by each type of operation (read, update, insert and

delete). Therefore, the authorization is controlled attribute by attribute and operation by operation on each attribute. This is the finest granularity that could be provided at the level of LMS. Delete is the only operation that cannot be executed on attribute basis but executed as an atomic action on all attributes.

#### 4.3.1.2.3 Adaptation process

The architecture of Business Schemas is flexible to allow the implementation of customizable FGACM. The FGACM to be implemented in each Business Entity are inferred by the Business Engine from Business Schemas, following the next rules:

##### LMS

If any of IForwardOnly, IScrollable, IRead, IUpdate, IInsert or IDelete interface is implemented, then an LMS must be instantiated. If IResult or ISet is implemented, then LMS cannot be instantiated.

##### Updatability

If any of IUpdate, IInsert IDelete or IResult is implemented, then LMS are instantiated as updatable. Otherwise, LMS are instantiated as read-only.

##### Scrollability

If the IForwardOnly interface is implemented, then LMS are instantiated as forward-only. If IScrollable is implemented, then LMS are instantiated as scrollable.

The adaptation process is accomplished using reflection on Business Schemas to analyze the implemented interfaces and the methods to be made available to be used in the Indirect Access Mode of CLI.

### 4.3.2 Policy Server

The Policy Server is responsible for storing metadata for FGACM and for informing other components about any modification. From DACC description, we see that there is no imposition to use any specific security policy. The only imposition is a security policy able to create permissions based on CRUD expressions executed on Business Schemas. From the main strategies for regulating access control policies, the use of RBAC to manage access control policies in a centralized way is widely accepted by RDBMS vendors, such as, for example, Microsoft SQL Server, Oracle and PostgreSQL. Thus, the choice for a security policy fell on an approach based on a RBAC policy. To that end, a basic security model was devised and is presented in Figure 43. This security model was devised to provide the basic mechanisms to support metadata of evolving FGACM. Thereby, important security issues for real applications, such as separation of duties and data abstraction are not addressed by this model. It may be used in real applications but that is not its goal. The main entities are: subjects (Sub\_Subject), sessions (Ses\_Session), client applications (App\_Application), roles (Rol\_Role), Business Schemas (Bus\_BusinesSchema), CRUD expressions (Crd\_Crud), authorizations (Aut\_Authorization) and delegations (Del\_Delegation). The correspondent logic model is presented in Annex A where a detailed description about the logic model is provided. Basically, a subject starts an application and a session is created. Then, the subject's roles are identified and the

correspondent granted permissions are identified and metadata of FGACM are sent to DACC. DACC dynamically built the FGACM. A role is activated if and only if:

- the role is assigned to an application that is also assigned to the subject;
- the subject has authorization to play the role or the role has been delegated to him. Roles are organized in general hierarchies to support the concept of multiple inheritance which promotes the ability to inherit permissions from several roles.

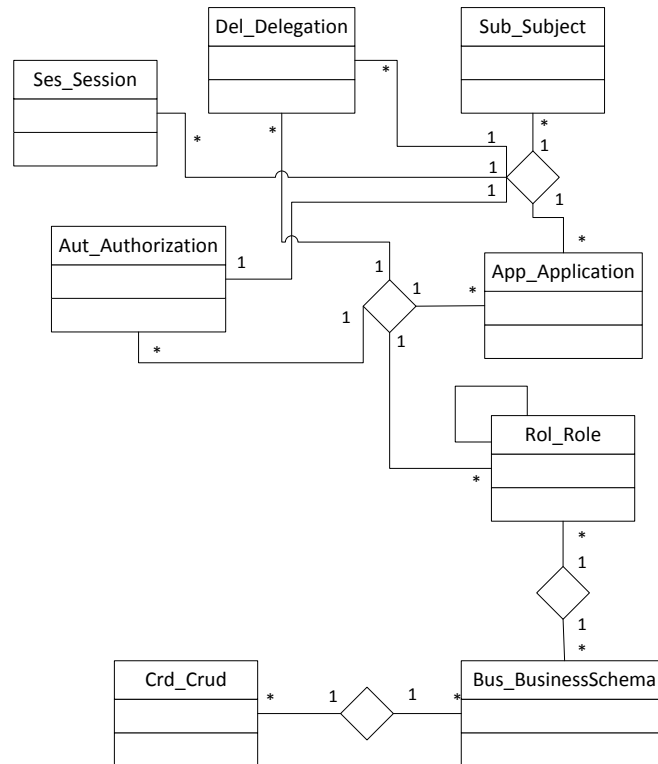


Figure 43. Access control Meta-model.

### 4.3.3 Policy Manager

The Policy Manager is a broker between the DACC and the Policy Server. Basically, the Policy Manager is responsible for sending to the DACC all the required metadata of FGACM to keep enforcement mechanisms aligned with the established FGACP. Now we discuss and present the methodology used to keep enforcing mechanisms updated with the established metadata, even when they evolve. The explanation is mainly based on Figure 37. Basically, the adaptation process has two moments: initialization and modification.

#### Initialization

Initialization is triggered when DACC start running. When DACC start running Business Logics are empty of Business Entities and CRUD expressions. Hence, in a first step, metadata of FGACM

need to be deployed to DACC. From them FGACM are automatically built, implemented and kept in Business Logic.

#### Modification

Modification is related to any modification in the metadata of FGACM kept by Policy Servers. Modifications in the metadata are captured by database triggers, which notify the Policy Watcher about the occurrence. The Policy Watcher becomes aware of the new state of the stored metadata and informs the Policy Manager. The Policy Manager checks all sessions (DACC) to be updated and, for each one, sends the correspondent metadata through the IAdaptation interface, see Figure 40. As a final note, DACC are dynamically adapted when modifications occur on delegations and on authorizations, which are the most frequent cases. If modifications occur in Business Schemas or CRUD expressions, they will only be reflected thereafter when their roles are assigned again.

## **4.4 Summary**

The DACA was presented and described in this chapter. The DACA comprises three main components: the Policy Server, the Policy Manager and the DACC. The Policy Server stores metadata of FGACM; the DACC is responsible for the dynamic implementation of FGACM in the client-side applications and the Policy Manager is a kind of proxy placed between the DACC and the Policy Server. Whenever the metadata of FGACM is updated, the DACA ensures that the correspondent mechanisms are automatically implemented in all running client-side systems. The DACA operation is split in three phases: the configuration phase to keep metadata of FGACM updated, the extraction phase to convey to programmers of application tiers a complete awareness about the implemented FGACM and, finally, the runtime phase where FGACM are dynamically built and kept updated in accordance with the established policies. Additionally, the services provided by the DACC are aligned with those provided by CLI conveying this way a similar user experience when compared with the CLI one. Another relevant aspect is the deployment process of CRUD expressions at runtime to DACC. This deployment process prevents programmers from writing CRUD expressions, which is in accordance with the previously announced security preoccupation.

The next chapter presents a proof of concept for the DACA, based on Java, JDBC, a RBAC model and a database relying on SQL Server.

## 5 Proof of Concept

This chapter presents a proof of concept based on Java, JDBC (sqljdbc4), Microsoft Northwind<sup>1</sup> database and using a RBAC policy. The proof of concept is available from here <sup>2</sup> and it is based on a scenario which intends to evaluate DACA against the research questions, identified in section 1.3. The main research question to be answered is: does DACA dynamically, at runtime, implement FGACM on business tiers and keep them updated when the policies evolve over time? If the answer is yes, then there are three additional second level research questions to be answered. The first one is related to security issues and stresses the need to evaluate if the DACA effectively controls the CRUD expressions being used. The second one is related to the possibility of providing a complete awareness about the established FGACM while programmers are writing source code for the application tiers of database applications. The third one is related to the possibility of keeping the advantages of CLI when they are used to enforce dynamic FGACM.

To answer these research questions some steps need to be accomplished. In a first step, it is necessary to build a platform based on DACA. The platform comprises all the basic components of the DACA and it is used to develop database applications based on DACA. Then, a database application based on the DACA is built. At this stage it is possible to answer the main research question and the first and second research questions of the second level. The third research question of the second level is partially answered but an additional step is needed to evaluate the decay of performance as consequence of the use of FGACM. To accomplish this task, a performance assessment is necessary to compare the responsiveness of solutions without access control and the responsiveness of the same solutions but now with access control based on the DACA. Both solutions must use standard CLI.

This chapter is organized as follows. In section 5.1 a platform based on the DACA is presented and the correspondent evaluation is made regarding the main question, the first second level question and part of the second question of the second level. In section 5.2, a performance assessment is carried out.

### 5.1 The DACA Platform

This section presents a platform based on the DACA. It is organized as follows: a scenario is presented to frame the context in which the proof of concept runs, then a proposal is presented for

---

<sup>1</sup> <http://www.microsoft.com/en-us/download/details.aspx?id=23654>

<sup>2</sup> Windows remote desktop connection - url: ned.av.it.pt; username: DACA; password: guest

the awarability of FGACM, then a proposal is presented for Policy Configurator (divided in two sub-sections) and then a database application based on the DACA is also presented.

### 5.1.1 Scenario

A scenario based on the DACA, beyond the constituent the DACA components, needs to provide a context from which the answers to research questions arise. Thus, beyond the constituent DACA components, the scenario also includes a database application. Some configuration is also defined, as seen in Figure 44.

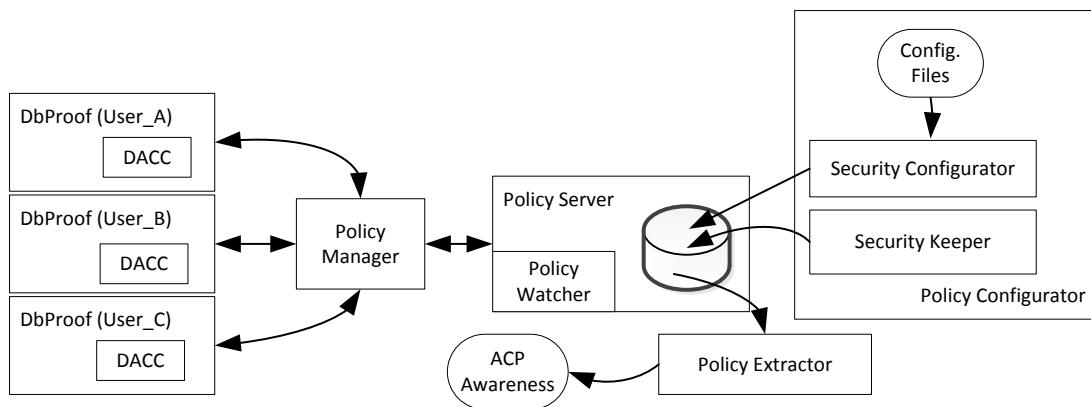


Figure 44. Block diagram for the proof of concept.

Next follows the description of the implemented scenario.

#### Policy Configurator

A simple Policy Configurator was built. It is responsible for defining the initial metadata for FGACM and also to enforce modifications in the metadata of FGACM while database applications are running. These aspects are essential to evaluate the DACA against the main research question and against the first research question of the second level. Two components were built: Security Configurator, to set the initial metadata, and Security Keeper to enforce modifications on metadata of FGACM.

#### Policy Extractor

A Policy Extractor was built to extract the necessary metadata for the building process of data structures responsible for the awareness context of the implemented FGACM. The collected results will answer the second research question of the second level.

#### Database application

A database application, herein known as DbProof, relying on Microsoft Northwind database, was built. DbProof uses a component based on the DACC to dynamically implement the established FGACM. Users are allowed to ask for the execution of CRUD expressions on Business Schemas. Whenever a permission is granted, CRUD expressions are executed, otherwise an error is raised. In order to visualize the granted permissions, an image similar to Figure 45 is



provided. Basically, whenever a modification in the metadata of FGACM is carried out through Security Keeper, the image will reflect the new set of permissions.

There are two points from which the main research question may be evaluated: interactively and through direct observation.

#### Interactively

Users can modify metadata of FGACM and then evaluate if their permissions have been updated in the DbProof. The feedback may be obtained by visualizing the state of permissions graphically presented on DbProof or users may try to execute CRUD expressions on Business Entity.

#### Direct observation

Users can modify metadata of FGACM and then visualize the contents of Business Logic. When the metadata is modified, the contents of Business Logic need to be in accordance with the established metadata.

A scenario was defined and built. It comprises several entities aimed at creating an environment where the DACA is evaluated. The main entities are:

#### Roles

Five roles were defined and organized in an hierarchical structure, as shown in Figure 45. The hierarchical structure is not essential for the thesis but it will provide feedback about propagation of permissions. A set of permissions is initially given to each role. This topic is described in the next paragraph.

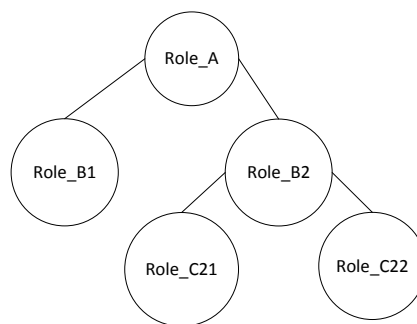


Figure 45. Hierarchy of roles.

#### Permissions

Permissions were defined and assigned to roles as shown in Table 4. A permission comprises a Business Entity that implements a Business Schema and the associated CRUD expressions. From Table 4 we see that there are five permissions each one identified by its Business Schema, ICat\_i, IPrd\_s, IPrdCat\_s, ICat\_s and ISup\_s, and the associated CRUD expressions. Each CRUD expression is identified by a unique identification (id) and also by a reference (Ref). These references are used to create the data structures for the awareness

environment about the implemented FGACM. All LMS, except ICat\_s, are read-only. ICat\_s is readable, insertable, updatable and deletable on all attributes.

Users

Three users were defined (user\_A, user\_B and user\_C) to evaluate if FGACM are built and kept updated by user. As each user may play different roles, FGACM need to be built and kept updated by the roles assigned to each user. By default, user\_A and user\_B play Role\_B2, Role\_C21 and Role\_C22. By default, user\_C play Role\_B1. Moreover, Role\_B1 may be assigned and unassigned by delegation to User\_A and User\_B, and may be assigned and unassigned to User\_C by authorization. Role\_B2, Role\_C21 and Role\_C22 may be assigned and unsigned to User\_A and User\_B by authorization and assigned and unassigned to User\_C by delegation.

Role	BS	CRUD		
		Id	Ref	Expression
Role_B1	ICat_i	1	all	Insert into Categories values(?,?,?)
Role_B2	IPrd_s	2	all	Select * from Products
		3	byId	Select * from Products where productId=?
		4	bySupplierId	Select * from Products where supplierId=?
	IPrdCat_s	5	byCategoryId	Select p.*, c.categoryName, c.Description from Products p, Categories c where p.CategoryID=c.CategoryID
Role_C21	ICat_s	6	all	Select * from Categories
		7	byId	Select * from Categories where categoryId=?
Role_C22	ISup_s	8	all	Select * from Suppliers
Role:	Role reference.			
BS:	Business Schema reference.			
Id:	CRUD identification.			
Ref:	CRUD reference.			
Expression:	CRUD expression.			

Table 4. Roles and the correspondent permissions for the implemented scenario.

**5.1.2 Awareness of FGACM**

In this subsection we discuss and present a solution to create in the Integrated Development Environment (IDE - NetBeans in our case) the data structures to convey a complete awareness of FGACM to programmers of applications tiers during the development process. The importance of this aspect is that programmers of application tiers can hardly master access control policies when schemas of databases and access control policies increase in complexity. This knowledge, when

integrated in the IDE, eases the development process of application tiers. In a first approach this stage seems useless because the DACC could eventually be automatically adapted to support all permissions and, therefore, to support all roles. Unfortunately, this approach cannot be followed because the DACC is agnostic regarding the adopted access control policy. DACC does not recognize the concept of roles. To overcome this difficulty, a tool, herein known as the Policy Extractor (see Figure 44), was designed to automatically extract and create data structures organized by roles and their permissions for the application under development. There were several options to formalize the data structures. Among them two were the favorite candidates: XML representation and object-oriented model representation. In spite of XML advantages, mainly for its technology independence, we chose to use the second option because it is much easier and faster to implement. The meta-data is built from the following data retrieved from Policy Servers:

- The supported roles from Rol\_Roles;
- The granted permissions for each role from Bus\_BusinessSchema and Crd\_Crud.

Figure 46 shows the data structures for Role\_B2. The data structures for Business Schemas are not shown but they are pure java interfaces easily inferred from Figure 41 and Figure 42. Basically, Role\_B2 explicitly provides permissions to use two Business Schemas, IPrd\_s and IPrdCat\_s and to execute CRUD expressions identified by 2, 3 and 4 on IPrd\_s and the CRUD expression identified by 5 on IPrdCat\_s. The names used to name roles, Business Schemas and identifications of CRUD expressions are retrieved from Rol\_Role, Bus\_BusinessService and Crd\_Crud, respectively. Moreover, inheritance is supported by supporting role hierarchies as foreseen by the RBAC. In case of Role\_B2, it inherits all permissions from Role\_C21 and Role\_C22 (Figure 46: line 11 – extends Role\_C21, Role\_C22). Additionally, programmers of business tiers do not have access to CRUD expressions, which is a key issue when the schemas of databases are themselves a part of the information to be protected.

The meta-data defined in Figure 46, in association with the architecture of the DACC, conveys to programmers of application tiers a complete awareness about the granted permissions for each role. As an example, Figure 47 shows the source-code for a subject playing the Role\_B2. From the selected role (Figure 47: line 41 - Role\_B2), programmers are statically driven to select one of the supported Business Schemas of Role\_B2 (Figure 47: line 41 – RoleB2.icat\_s). Then, programmers are semantically oriented to select one of the CRUD expressions supported by the selected Business Schema (Figure 47: line 41 - Role\_B2.icat\_s\_byId). Any security violation at the level of Business Schemas is checked at development time and source-code will not compile if some security violation is detected. After being deployed, FGACP may evolve and an exception is raised if, for some security reason, this role is not assigned anymore. The selected CRUD expression is executed and one runtime parameter is used (Figure 47: line 43). If a row has been selected (Figure 47: line 44), programmers can choose any action supported by the Business Schema to access the contents of LMS (pop-up window partially shows all actions, from Figure 47: line 45). This pop-up window also conveys to programmers complete and type-safe awareness about the actions supported on LMS of ICat\_s, independently from the CRUD expression being executed. The pop-window shows that subjects playing Role\_B2 can read, update and insert all attributes of table Categories. To completely separate methods from IRead, IUpdate and IInsert, we suggest the use of a unique prefix: 'r', 'u' and 'i', respectively, for each method.

```

11 public interface Role_B2 extends Role_C21, Role_C22 {
12     public static final Class<IPrd_s> iprd_s=IPrd_s.class;
13     public static final int iprd_s_all=2;
14     public static final int iprd_s_byId=3;
15     public static final int iprd_s_bySupplierId=4;
16     public static final Class<IPrdCat_s> iprdcat_s=IPrdCat_s.class;
17     public static final int iprdcat_s_byCategoryId=5;
18 }
    
```

Figure 46. Role\_B2 definition.

```

39     ICat_s c=null;
40     try{
41         c=session.businessEntity(Role_B2.icat_s, Role_B2.icat_s_byId);
42     } catch(DACA_Exception ex) { /* code */ }
43     c.execute(categoryId);
44     if (c.moveToNext()) {
45         c.
46     }
47     }
48     }
49     }
50     }
51     }
52     }
53     }
54     }
55     }
56     }
57     }
58     }
    
```




Figure 47. Programmers awareness about FGACM for Role\_B2.

### 5.1.3 Security Configurator

In this implementation, the Security Configurator has no GUI. It is a component that reads metadata from software artifacts (classes and interfaces) to partially fill the Policy Server with the needed metadata for one database application. We have not addressed the configuration process of subjects because we consider that the new key concept introduced by the DACA, that deserves more attention, is the concept of permission which is based on Business Schemas and on CRUD expressions. Next follows the main software artifacts that were developed for the implemented scenario.

#### IApplication

IApplication, see Figure 48, defines the application name and the implemented root roles. In this scenario there is only one root role: Role\_A.

```

13 public interface IApplication {
14     String app = "app";
15     Class[] roles = new Class[] {
16         IRole_A.class};
17 }

```

Figure 48. Application definition.

### Role B2

IRole\_B2, see Figure 49, defines all Business Schemas supported by Role\_B2: IPrd\_s and IPrdCat\_s. Additionally, Role\_B2 extends Role\_C21 (IRole\_C21) and Role\_C22 (IRole\_C22). The definition of the remaining roles follows a similar approach.

```

12 public interface IRole_B2 extends IRole_C21, IRole_C22 {
13     Class[] role_b2=new Class[] { IPrd_s.class,
14                                     IPrdCat_s.class };
15 }

```

Figure 49. Role\_B2 definition.

### IPrd\_s

IPrd\_s, see Figure 50, defines the Business Schema IPrd\_s in terms of the implemented interfaces and in terms of the supported CRUD expressions. IPrd\_s implements LMS (IExecute, IScrollable and IRead) and supports CRUD expressions 2, 3 and 4. The definition of the remaining Business Schemas follows a similar approach.

```

11 public interface IPrd_s extends IExecute, IScrollable, IRead {
12     String cruds[] = new String[] {ICrud.sPrd_all,
13                                     ICrud.sPrd_byId,
14                                     ICrud.sPrd_bySupplierId};
15 }

```

Figure 50. Business Schema IPrd\_s definition.

### IRead

IRead, see Figure 51, defines the getter methods to read data from the LMS in accordance with the established FGACP for IPrds\_s. In this case all attributes are readable.

```

13 public interface IRead {
14     int rProductId() throws SQLException;
15     String rProductName() throws SQLException;
16     int rSupplierId() throws SQLException;
17     int rCategoryId() throws SQLException;
18     String rQuantityPerUnit() throws SQLException;
19     float rUnitPrice() throws SQLException;
20     int rUnitsInStock() throws SQLException;
21     int rUnitsOnOrder() throws SQLException;
22     int rRecorderLevel() throws SQLException;
23     boolean rDiscontinued() throws SQLException;
24 }

```

Figure 51. IRead definition.

ICrud

ICrud, see Figure 52, defines the CRUD expressions to be supported by all Business Schemas. Each CRUD expression is identified by its name (String variable), which has been defined in Table 4 as being the references of CRUD expressions.

```

11 public interface ICrud {
12     String iCat_all="Insert into Categories values (?, ?, ?)";
13     String sPrd_all="Select * from Products";
14     String sPrd_byId="Select * from Products where ProductId=?";
15     String sPrd_bySupplierId="Select * from Products where supplierId=?";
16     String sPrdCat_byCategoryId="Select p.*,c.categoryName,c.Description " +
17         "from Products p, Categories c " +
18         "where p.CategoryID=c.CategoryID";
19     String sCat_all="Select * from Categories";
20     String sCat_byId="Select * from Categories where categoryId=?";
21     String sSup_all="Select * from Suppliers";
22 }
    
```

Figure 52. Definition of all CRUD expressions.

Based on this information, Security Configurator automatically fills: App\_Application, AppRol, Rol\_Role, RolBus, Bus\_BusinessSchema, BusCrd, Crd\_Crud and also Aut\_Authorization (set to yes by default). AppRol, RolBus, BusCrd are not represented in Figure 43. These tables are used to decompose M:N relationships between tables identified by their prefixes.

**5.1.4 Security Keeper**

The Security Keeper (part of Policy Configurator, see Figure 44) was designed to ease the process of modifying, at runtime, the granted roles to users which are stored in the Policy Server. To change role assignment for each one of the three users, a simple GUI application is used, see Figure 53. Basically roles are automatically granted and denied by checking and unchecking, respectively, the shown check boxes for each user, in accordance with the established scenario.

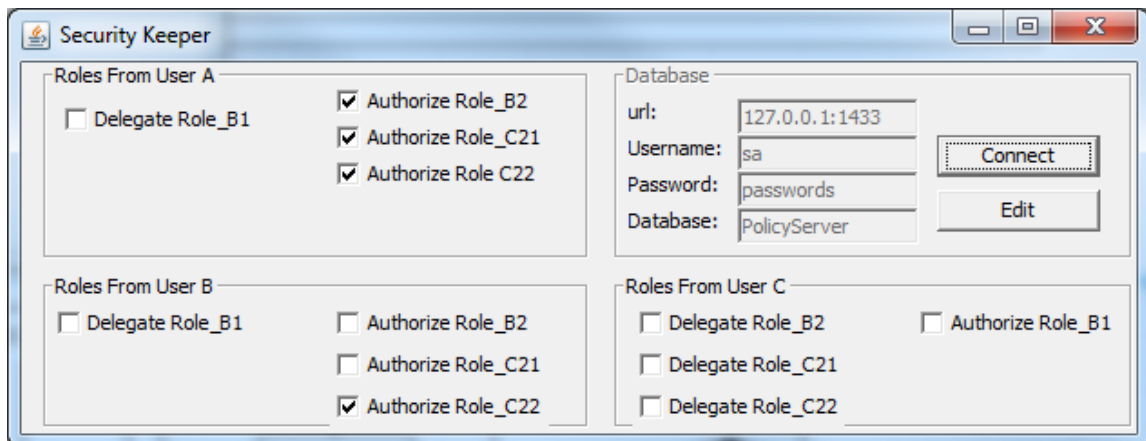


Figure 53. Security keeper.

### 5.1.5 DbProof

The final component is DbProof, see Figure 54. DbProof is a database application based on the DACA. The business logic is dynamically built and updated at runtime from established metadata of FGACM. The DbProof and the Security Keeper are simultaneously used to assess if database applications based on the DACA effectively, in real situations, keep FGACM aligned with evolving FGACP. From the DbProof, we may choose one of the three supported users. Then Business Schemas and CRUD expressions are selected to be executed one at a time. If permission is granted, the CRUD expression is executed, otherwise an error message is shown. Figure 54 is the default context of user A. Green circles (lighter gray for black and white prints) are for granted roles and red circles (darker gray for black and white prints) are for denied roles. The colors are updated whenever the assignment state of roles is modified. It also shows that Business Schema ICat\_s and CRUD expression (allFromCategories) are selected and have been already executed. ILMS of ICat\_s

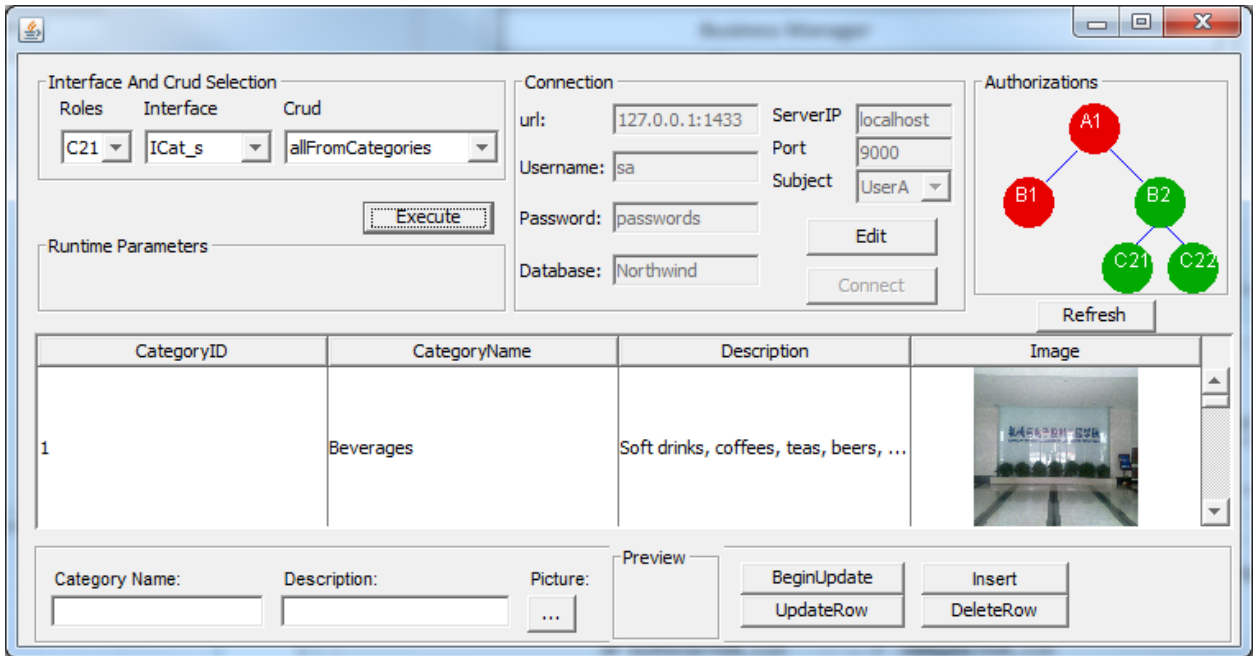


Figure 54. DbProof.

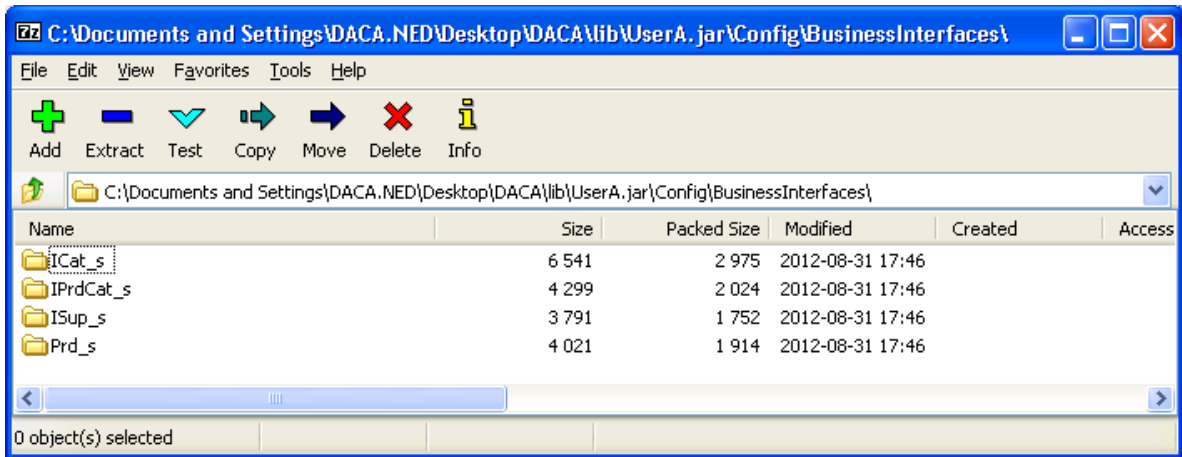


Figure 55. Business Schemas implemented for user User\_A.

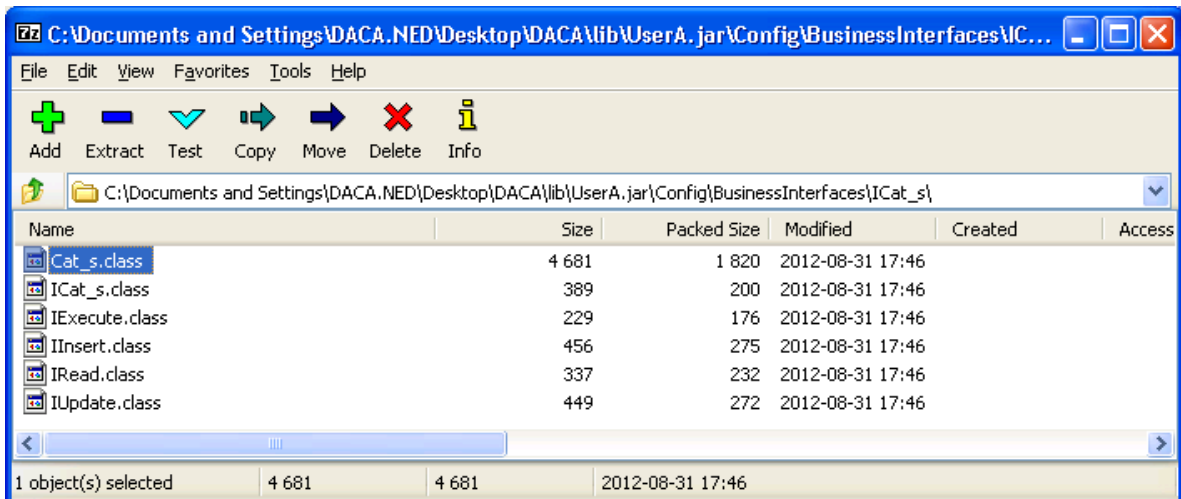


Figure 56. LMS interfaces for Cat\_s Business Schema.

is also updatable, insertable and deletable and, therefore, some additional actions are available at the bottom of the GUI. The dynamic enforcement of FGACM is directly observable using a common unzip tool to analyze the contents of the files (Business Logic - Jar file) containing the Business Entities. Business Entities are inserted and removed in accordance with the granted and denied roles to users. Figure 55 partially presents the contents for the Business Logic belonging to User\_A when roles Role\_B2, Role\_C21 and Role\_C22 are granted. Business Logic contains four folders, each one for each Business Schema: ICat\_s, IPrdCat\_s, ISup\_s and IPrd\_s. Figure 56 presents the LMS interfaces of Business Schema ICat\_s and also the correspondent Business Entity, Cat\_s. The remaining interface, IScrollable, is used from a pool shared by all Business Schemas and, as such, it is not present in this folder.

## 5.2 Performance Assessment

The performance assessment is focused on evaluating and comparing the performance of the DACC based on the DACA and the performance of solutions based on a standard CLI API and without any access control mechanism. Java, JDBC and SQL Server 2008 have been chosen as the basic core technologies to support the assessment. The test-bed relies on a PC Asus-P5K-VM, Intel Duo Core E6550 @2.33 GHz, 4.00 GB RAM, Windows XP Professional Service Pack 3, Java SE 7 (1.7.0\_22-b13), JDBC(sqljdbc4) and SQL Server 2008. In order to promote an ideal environment, the following actions were taken:

- The running threads were given the highest priority;
- All non-essential processes/services were cancelled;
- A new database was created for the running tests;
- For each individual measurement a new table with a different name is created from scratch to avoid SQL Server to take advantage of any optimization process;
- Some default SQL Server database properties were changed, such as Auto Update Statistics = false and Recovery Model = Simple, to minimize its overhead.



## 5.2.1 Methodology

The methodology followed to collect the needed measurements was based on measuring how long a task takes to execute. To achieve this goal, the method *system.nanoTime()* was used. In spite of being a very easy methodology to collect measurements, it conveys an error during the measurement process. To evaluate the impact of the act of measuring, the collected values showed that the impact is always under 310ns and that the minimum clock tick is 1ns. From these values, and in order to ensure that errors were always below 1%, all measurements associated with the performance assessment were collected with a minimum time span of 31,000ns. In several situations it was necessary to repeat the same code as often as necessary to get a minimum of 31,000ns. To avoid additional errors with the repeating process, the code was sequentially repeated and not iteratively repeated. Table 5 presents the general strategy followed to collect and compute each measurement.

```

1 repeat 5 rounds
1.2 get a new container to keep the collected measurements
1.3 prepare initial conditions
1.4 repeat: 100 cycles
1.4.1 start timer
1.4.2 run scripts (must take at least 31,000ns)
1.4.3 stop timer
1.4.4 keep elapsed time if it is one of the 5 best in this cycle
1.4.5 release all unnecessary objects
1.4.6 garbage collector activation
1.4.7 sleep 100ms (other system processes may need to run)
2 keep the best average time of the 5 rounds

```

Table 5. Strategy to collect and compute measurements.

From the performance assessment point of view, DACC may be split into two main phases: the creation phase and the execution phase. The creation phase is related to activities that have no equivalent on standard CLI. The execution phase comprises the activities that are shared by standard CLI and by DACC.

### Creation phase

This creation phase comprises activities such as the instantiation of DACC, building process of Business Entities and instantiation of Business Sessions and Business Entities. All activities, except the instantiation of Business Sessions and Business Entities occur only once or very sparsely and therefore their impact has not been considered to be evaluated in this research. On the other hand, instantiation of Business Sessions and Business Entities occur very frequently conveying an overhead the impact of which must be evaluated. The collected measurements were obtained using a case study based on a scrollable and updatable LMS. This LMS type is the one that comprises more methods and more data structures. IRead, IUpdate and IInsert were defined with 25 methods each. In spite of not being a critical aspect, it is expectable that the complexity of this Business Schema will be above most of the real Business Schemas. The collected time to instantiate a Business Session (TBS) and a Business Entity (TBW) is presented in Table 6 a). Instances of Business Entities are herein known as Business Workers.

### Execution phase

The execution phase is mainly focused on evaluating the overhead induced by the invocation of

Business Entities' methods. Two main approaches may be followed to carry out the performance assessment:

- Use a DACC and assess it against a standard JDBC component based on scenarios and case studies;
- Develop a general environment to evaluate the overhead induced by the wrapping process implemented by each method of each Business Entity.

After some reflection, it came clear that the latter approach would bring a significant advantage over the former approach. Methods of Business Entities are general and not tied to any particular use case. Moreover, their use and their functionalities are clearly stated, leading to the possibility of developing a mathematical model to express and evaluate its impact on any possible scenario: the overhead is only dependent on the additional time to call the wrapping method. Thus, if the overhead is known for all methods of Business Entities in a running context (CPU, Operating system, etc.), it will be possible to mathematically compute the induced overhead for any Business Entity running on that context.

The activities related to the execution phase are basically the invocation of Business Entities methods. Each Business Entity method wraps a block of code of the standard CLI. Thus, the overhead may be measured by evaluating the time to execute the additional code when using a Business Entity method. To this end, we introduce the concept of reduced method signature (RMS). RMS derives from the widespread concept of method signature but it does not include the method name. All methods of Business Entities are classified in two different groups: methods with a fixed RMS and methods with a variable RMS. Methods with a fixed RMS are, by far, the major group. The only method that does not have a fixed RMS is *execute* with parameters. In order to predict the overhead induced by every wrapping method, it was decided to measure the finest grain overhead induced by each possible variation in RMS. Two examples: measure the induced overhead by each additional argument of any data type and measure the induced overhead by returning any data type. To achieve this goal, two types of measurements were collected as shown in Table 6 b).  $TR_i$  are the collected measurements for methods with no arguments and returning the data types shown in the column *Data type*. Examples: *void m1()* and *int m2()*.  $TA_i$  are the collected measurements for calling a method with 10 arguments of type *Data type* and returning *void*. The contribution of each individual argument is computed as  $(collectedMeasurement - TR_1)/10$ . This approach was validated by carrying out some additional tests using less than 10 and more than 10 arguments and with and without a returning value. From Table 6 it is possible to compute the absolute overhead induced by any RMS running on the same context and, therefore, of any method of each Business Entity.

In spite of being important, the data shown in Table 6 do not give any insight about their

TBS	TBW	Data type	TR <sub>i</sub>	TA <sub>i</sub>	I	Data type	TR <sub>i</sub>	TA <sub>i</sub>	i
1023	387	void	14	-	1	string	27	98	6
		byte	26	89	2	float	45	132	7
		short	26	114	3	double	45	132	8
		int	26	91	4	boolean	26	89	9
		long	42	129	5	char	28	103	10

Table 6. Collected measurements for a) TBS, TBW and for b) RAM in ns.

	Description	Algorithm	Typical DACA usage	Overheads
$SS_i(R,A,L)$	Assess $SS_i$ . All attributes from all rows are read from the LMS.	Create new table(*) Insert R rows Start timer Select all rows For each row Read all attributes Stop timer	[SQL: Select * from table] // create business session // instantiation of business worker a a.execute(); while ( a.moveToNext() ) { idl = a.id1(); // more attributes } //release business session	TBS+ TBW+ TR <sub>1</sub> + (TR <sub>9</sub> + TR <sub>4</sub> *nAtt) *nRows
$SS_i(R,A,(FoUp,ScUp))$	Assess $SS_i$ . All attributes of all rows are set one by one in the LMS and committed to the database.	Create new table(*) Start timer Select all(0) rows For each new row Insert all attributes Commit Stop timer	[SQL: Select * from table] // create business session // instantiation of business worker a a.execute(); while ( a.moveToNext() ) { a.beginUpdate(); a.id1(id1); // more attributes a.updateRow(); } //release business session	TBS+ TBW+ TR <sub>1</sub> + (TR <sub>9</sub> + TR <sub>1</sub> + (TR <sub>1</sub> +TA <sub>4</sub> ) *nAtt+ TR <sub>1</sub> ) *nRows
$SS_d(R,A,(FoUp,ScUp))$	Assess $SS_d$ . All attributes of all rows are updated one by one in the LMS and committed to the database.	Create new table(*) Insert R rows Start timer Select all rows For each row Update all attributes Commit Stop timer	[SQL: Select * from table] // create business session // instantiation of business worker a a.execute(); while ( a.moveToNext() ) { a.beginInsert(); a.id(id); // more attributes a.insertRow(); } //release business session	TBS+ TBW+ TR <sub>1</sub> + (TR <sub>9</sub> + TR <sub>1</sub> + (TR <sub>1</sub> +TA <sub>4</sub> ) *nAtt+ TR <sub>1</sub> ) *nRows
$SS_d(R,A,(FoUp,ScUp))$	Assess $SS_d$ . All rows are deleted one by one from the LMS and committed to the database.	Create new table(*) Insert R rows Start timer Select all rows For each row Delete row Commit Stop timer	[SQL: Select * from table] // create business session // instantiation of business worker a a.execute(); while ( a.moveToNext() ) { a.deleteRow(); } //release business session	TBS+ TBW+ TR <sub>1</sub> + (TR <sub>9</sub> + TR <sub>1</sub> +) *nRows

Note: (\*) in case of join, the second table is also created with 5 attributes and with the same number of rows as the main table.

Table 7. Scenarios for the Select expression: algorithms and typical component usage.

relative impact on real cases. The impossibility to assess all cases led to a survey to define some scenarios that could be representative of common situations and, above all, that could give a perspective about DACC behaviour regarding its impact on the overall performance. To this end, we needed to identify the relevant aspects directly related and controlled by application tiers that could influence a business tier performance based on DACA. Based both on empirical experiences and on knowledge about DACC, the aspects considered relevant (and confirmed in Figure 57, Figure 58 and Figure 59) were: types of CRUD expressions (Select, Insert, Update, Delete), types of LMS (Forward-Only and Read-only (FR), Forward-only and Updatable (FU), Scrollable and Read-only

	Description	Algorithm	Typical DACA usage	Overheads
SI	Assess Rows are inserted one by one in the database through the execution of a parameterized Insert statement.	Create a new table Start timer Create statement for each row insert all attributes Stop timer	[SQL: Insert into table values (att...)] // create business session // instantiation of business worker <i>b</i> <i>b.execute(att1, att2, ..., attn.);</i> // more inserts //release business session	TBS+ TBW+ (TR <sub>1</sub> +TA <sub>4</sub> ) * nAtt*nRows
SU	Assess Rows are updated one by one in the database through the execution of a parameterized Update statement.	Create a new table Insert R rows Start timer Create statement For each row Update all attributes Stop timer	[SQL: update table set (...) where pk=?] // create business session // instantiation of business worker <i>c</i> <i>c.execute(pk,att2,...,attn);</i> // more updates //release business session	TBS+ TBW+ (TR <sub>1</sub> +TA <sub>4</sub> ) * nAtt*nRows
SD	Assess Rows are deleted one by one through the execution of a parameterized Delete statement.	Create a new table Insert R rows Start timer Create statement For each row Delete row Stop timer	[SQL: delete from table where pk=?] // create business session // instantiation of business work <i>d</i> <i>d.execute(pk);</i> // more deletes //release business session	TBS+ TBW+ (TR <sub>1</sub> +TA <sub>4</sub> )  *nRows

Table 8. Scenarios for the Insert, Update and Delete expressions: algorithms and typical component usage.

(SR), Scrollable and Updatable (SU)), the number of rows to be processed, the number of attributes of each row and the query complexity. Tests with pre-compiled and compiled-on-the-fly CRUD expressions were also carried out. Only measurements relative to the pre-compiled CRUD expressions will be presented because the collected results with compiled-on-the-fly CRUD expressions were so close that their presentation would not bring any novelty to the final conclusions. Exogenous aspects such as hardware architecture, hardware components, operating systems, database servers, middleware software and communication infrastructures were not considered because their impact is not directly or indirectly dependent on DACC. To address all the presented aspects, seven main scenarios were defined as presented and described in Table 7: *SS<sub>r</sub>*, *SS<sub>i</sub>*, *SS<sub>u</sub>*, *SS<sub>d</sub>*, *SI*, *SU* and *SD*. Then, for each main scenario, two facets were created to handle different number of rows,  $R \in \{1, 5, 10, 25, 50, 100, 250, 500, 750, 1000, 1250, 1500, 1750, 2000\}$ , and tables with different number of attributes,  $A \in \{5, 10, 20\}$ . These facets were defined from empirical experiences that were carried out to delimit the range of values that could forecast the behaviour of other scenarios. Additionally, to have an idea about the impact of increasing queries complexity for the *SS<sub>r</sub>*, two queries were defined: one with no joins (NJ - select table.\* from table) and another with a simple join (on their primary keys) comprising two tables (WJ - Select table.\* from table, table1 where table.id=table1.id). In this case table1 had a fixed number of 5 attributes. The attributes of all tables were all defined as being of type *integer* and *not null*. They could be of any other data type or

any combination of several data types. There was no reason to accept or refuse any possibility. All tables are created with a primary key on the first attribute. A formalization of a general scenario may be expressed as  $S(R,A,L)$  where  $S$  is the scenario,  $A$  the number of attributes,  $R$  the number of rows and  $L$  the type of LMS.

For each scenario, Table 7 and Table 8 present a concise description for the algorithm used to collect measurements, the source code for the DACC and, finally, the total absolute overhead induced by each method (nAtt is for the number of attributes, nRows for the number of rows). Check Table 6 to remember the meanings of acronyms used in column *Overheads* on Table 7 and Table 8.

## 5.2.2 Collected Results

Figure 57, Figure 58 and Figure 59 present the graphics for all scenarios. Three types of information were selected to be presented: the induced % overhead by DACC, the absolute induced overhead by DACC (in milliseconds) and the % contribution of each component to the total % induced overhead (CBS for TBS, CBW for TBW and CSR for the execution phase). The importance of this information is: 1) the induced overhead cannot be completely understood if only the % values or only the absolute values are given. They complement each other. 2) The overhead analysis in components give an insight about its composition opening the opportunity to evaluate the possibility of taking measures to lessen its impact. Graphics for components overheads do not present the range 150 till 2000 rows because CSR is practically the only relevant component in that range, as it may be easily inferred. Additionally, in these graphics, the number of rows is clustered by the number of attributes.

To completely understand the presented results, some additional information about the collected measurements is essential:

- Performance decreases from forward-only (Fo) to scrollable (Sc) and from read-only (Ro) to updatable (Up) LMS. This derives from the fact that database servers create server cursors, for other LMS than FoRo, with increased management complexity to control client operations on LMS.
- Performance increases (number of selected rows/second) when the number of selected rows increases.
- Performance decreases (number of selected rows/second) when the number of attributes increases.
- Performance decreases (number of selected rows/second) when select statements include a join.

Another relevant issue is the fact that the absolute overhead value has been formalized for each scenario, see Table 7 column *Overheads*. It does not depend on the LMS type even not on the query complexity: it only depends on the methods used during the execution phase.

To discuss the collected measurements for each scenario, the scenarios were aggregated in three main groups: 1) SS<sub>r</sub>; 2) SS<sub>u</sub>, SS<sub>i</sub> SS<sub>d</sub> and 3) SI, SU and SD.

### 5.2.2.1 Scenario SS<sub>r</sub>

The graphics for SS<sub>r</sub> are shown in Figure 57. Figure 57 a), b), c) and d) show the % overhead. Columns are for Select expressions with no join (NJ) and lines are for Select expressions with a join (WJ). Each Select expression is executed on tables with 5, 10 and 20 attributes. The behaviour shows that the % overhead increases from ScUp->FoUp->ScRo->FoRo, when the number of rows increases, when the number of attributes increases and when Select expressions do not include a join. The % overhead is minimum for very few number of rows but it may rise till 7% for SS<sub>r</sub> (2000, 20, FoRo) with no join. We may conclude that the percentage impact of DACC may not be negligible for some marginal SS<sub>r</sub>, mainly for FoRo LMS with thousands of rows, conveying the need to proceed with a previous assessment. Figure 57 e) shows the absolute overhead. It increases with the number of rows and with the number of attributes reaching about 1.1ms for 2,000 rows and 20 attributes.

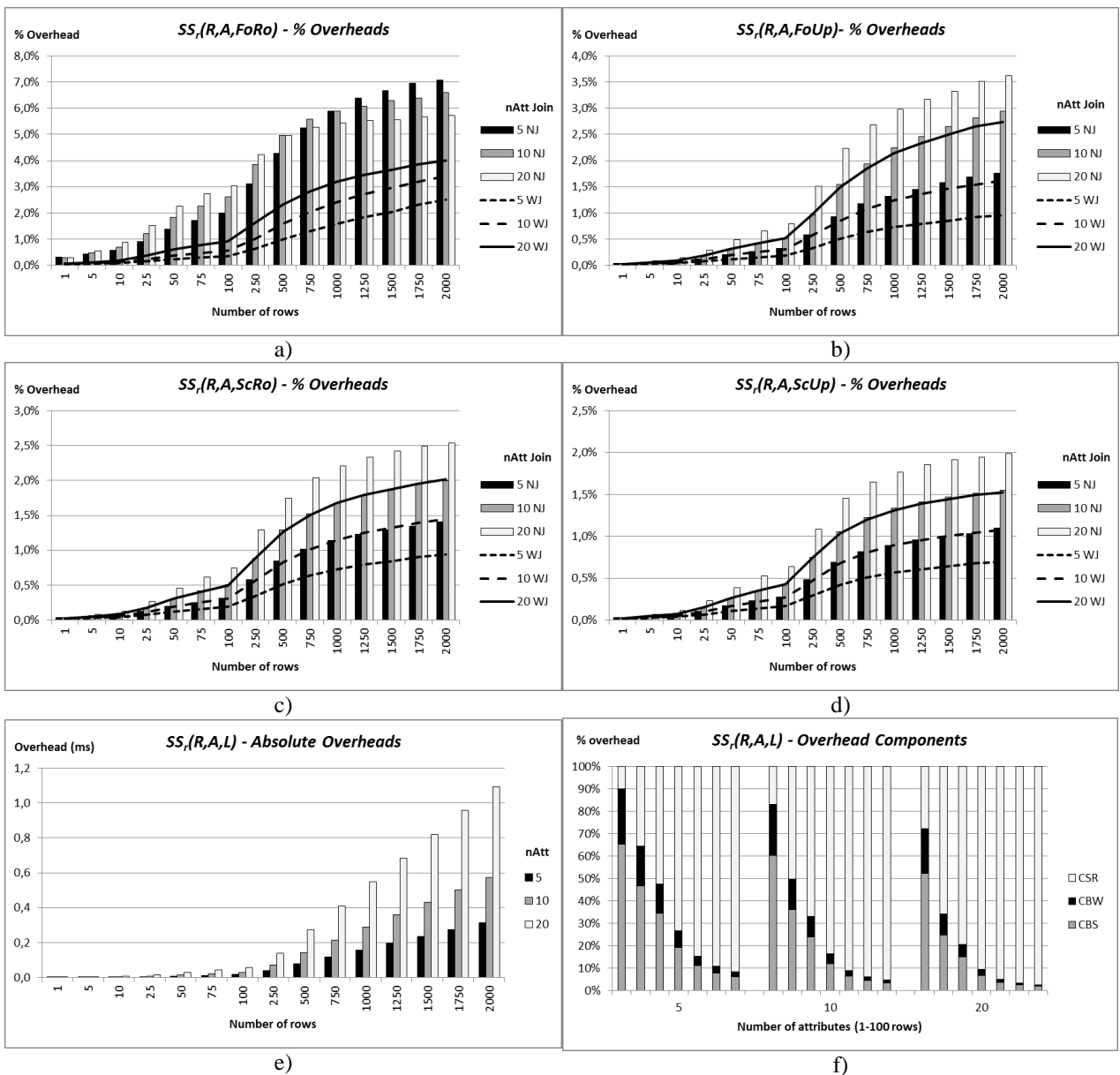


Figure 57. Graphics for scenario SS<sub>r</sub>.

Figure 57 f) shows the overhead components. In critical situations, when the overhead must be minimized, programmers may also use the individual components. The same instance of Business Sessions and Business Workers may very often be used over and over, this way avoiding its correspondent overhead. For example, with 5 attributes the overhead may be reduced from 90% for 1 row till 50% for 10 rows, which may be considered very significant.

### 5.2.2.2 Scenarios $SS_i$ , $SS_u$ and $SS_d$

The graphics for these scenarios are shown in Figure 58. Figure 58 a), c) and e) show the absolute (columns) and the percentage overhead (lines). The lines represent the combination between each LMS type (only FoUp and ScUp are allowed) and each possible number of attributes. The general behaviour

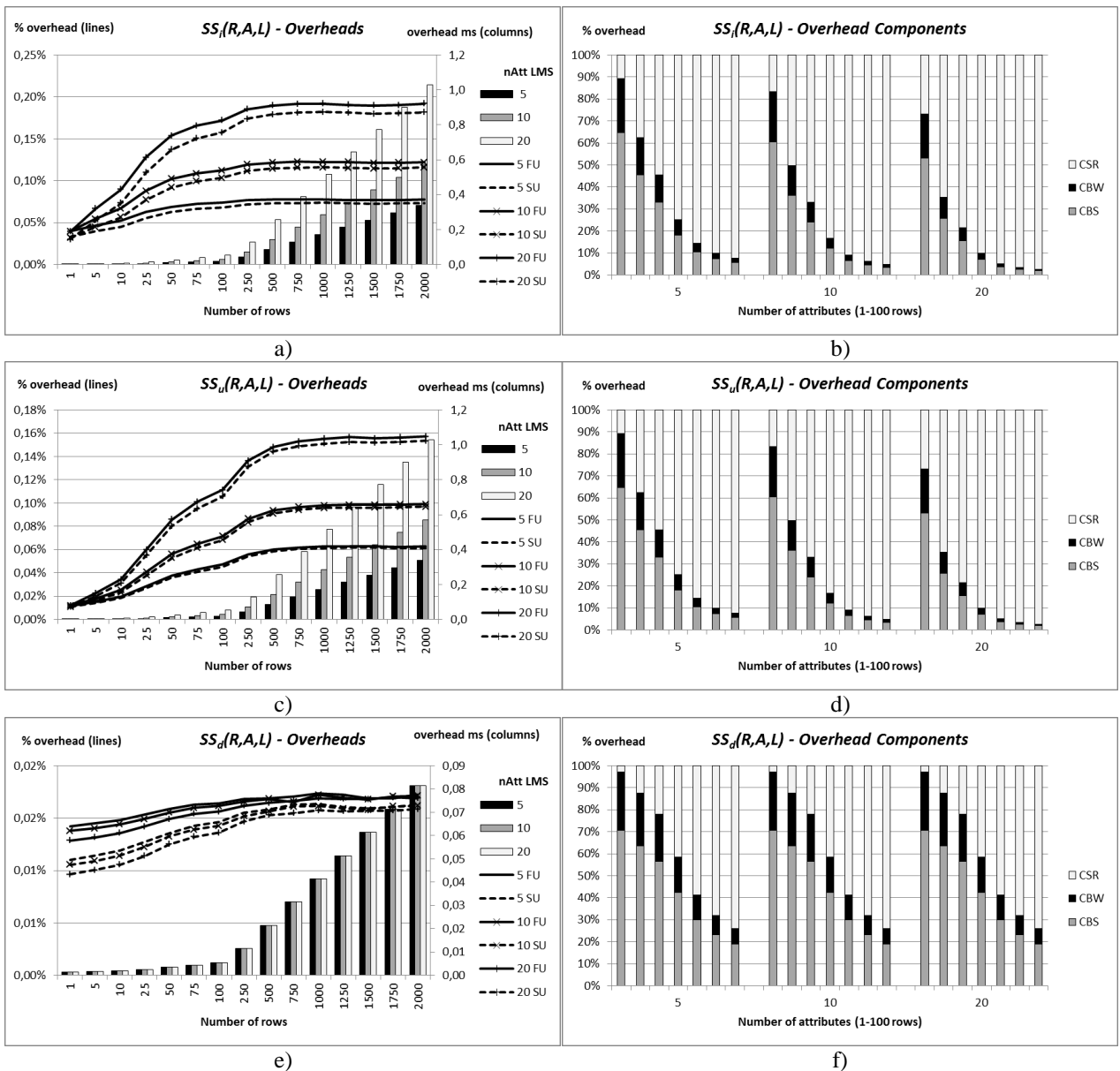


Figure 58. Graphics for scenarios  $SS_i$ ,  $SS_u$  and  $SS_d$ .

shows that: 1) the percentage overhead is practically independent of the LMS type; 2) the number of attributes is only relevant in  $SS_i$  and  $SS_u$ ; 3) all percentage overheads tend to be stabilized when the number of rows reaches a certain threshold; 4) the maximum percentage overhead is much lower than in  $SS_r$ : maximum 0.2%, 0.16% and 0.02% for  $SS_i$ ,  $SS_u$  and  $SS_d$ , respectively; 4) as expected, the absolute overhead raises with the number of rows and with the number of attributes. In spite of its low impact, programmers may still act at the level of the overhead components, see Figure 58 b), d) and f). CBS and CBW together, for low number of rows, spend most of the overhead, indicating that the overhead, in this range, may be practically eliminated for all  $SS_i$ ,  $SS_u$  and  $SS_d$ .

### 5.2.2.3 Scenarios SU, SI and SD

The graphics for these scenarios are shown in Figure 59. In these scenarios, CRUD expressions (insert, update, delete) are not executed through an LMS but directly on the database server

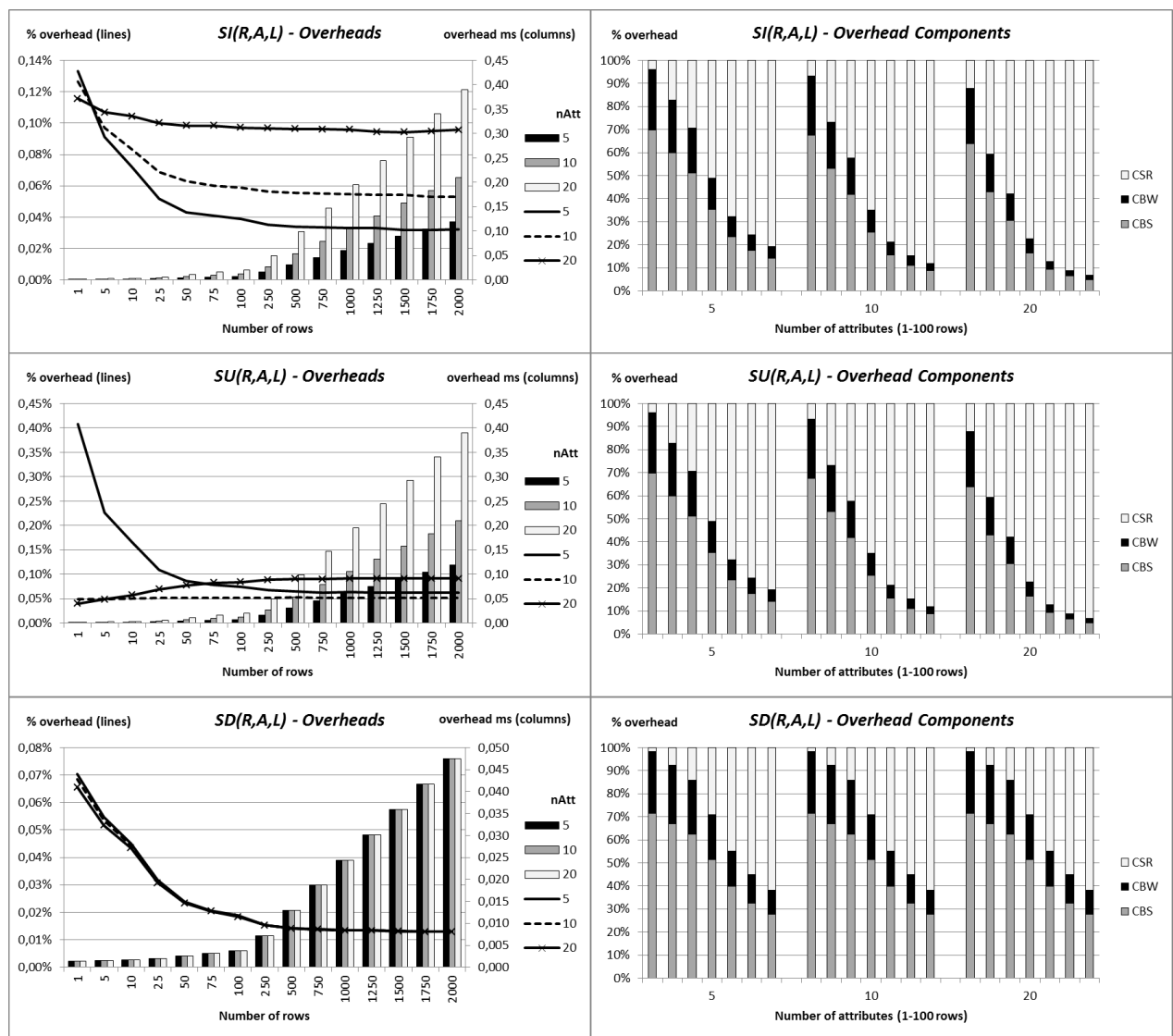


Figure 59. Graphic for scenarios SI, SU and SD.



through parameterized CRUD expressions. Figure 59 a), c) and e) show the absolute (columns) and the % overheads (lines). The general behaviour is not aligned with the previous ones. The % overhead does not increase when the number of rows increases but converges to a constant value in each scenario. Collected measurements range from 0.3% till 0.03%. In critical situations, the maximum values may be reduced from 75% till 98% if CBS and CBW are carefully used, see Figure 59 b), d) and f).

## 5.3 Results Evaluation

This section is focused on evaluating the obtained results to verify how the research questions have been fulfilled. The main research question is verified at the first place and then the second level research questions are also verified.

### 5.3.1 Dynamic FGACM on business tiers

The main research question to be answered was defined as: *“is it possible to dynamically, at runtime, implement FGACM on business tiers and keep them updated when the policies evolve over time?”*

To answer this question a platform was devised. It comprises two main components: a server component where metadata of FGACM are stored and kept updated and a client component deployed in every client system responsible for implementing the FGACM at runtime. To evaluate if FGACM are dynamically implemented and updated at runtime a platform based on the DACA was built and a scenario was defined and implemented. The scenario included three users and a set of hierarchized roles and their associated permissions. The scenario provided a tool to allow the dynamic modification of assigned roles to each individual user. Whenever a role was assigned or unassigned, it was confirmed that FGACM are dynamically updated at runtime at the client side systems. The confirmation was verified in three different ways:

- 1) The devised DbProof component explicitly shows for each user, through a graphic, the hierarchized roles and their assignment state, see Figure 54. The assignment state indicates for each role if a role is assigned or unassigned. It was confirmed that the assignment state was updated whenever a modification was enforced at the level of the metadata of FAGCM.
- 2) The DbProof provides an interface where permissions (Business Entities and CRUD expressions) are selected to be executed, see Figure 54. It was confirmed that the success to execute any permission was always in accordance with the state assignment of each role.
- 3) A material verification was also conducted. The material verification consists in looking inside the Business Logic to check its contents. The contents of Business Logic of each user was inspected using a common unzip tool, see Figure 55 and Figure 56. The contents of each Business Logic confirmed that it was in accordance with the assigned permissions.

Before these multi-verifications, there no is doubt that DACA positively answers the main research question of this thesis.

### 5.3.2 Security

This sub-section evaluates how the first research question of the second level is answered by the DACA. The research question is: *“Is there any possibility to supervise the use of CRUD expressions effectively when protected data is being access to protected data?”*. As previously shown, current approaches allow users to write any CRUD expression freely. The endless expressiveness of the SQL language opens the possibility to existence of security gaps. In order to overcome this situation, the DACA does not allow users to write any CRUD expression. Instead of writing CRUD expressions, users are only allowed to use CRUD expressions that are defined by security experts and put at their disposal by the DACA, see Figure 46 and Figure 47. Additionally, the DACA has the ability to identify the permissions granted to each user and make them dynamically available to be used. This way, users have no possibility to issue their own CRUD expressions.

### 5.3.3 FGACM awareness

This sub-section evaluates how the second research question of the second level is answered by the DACA. The research question is: *“Is it possible to overcome this difficulty by providing programmers with a complete awareness about the established FGACM?”*. Current approaches do not give any guidance on the established FGACP neither on the implemented FGACM.

To answer this question, a tool was devised to specifically address this issue – Policy Extractor, see Figure 37. Policy Extractor reads the metadata of FGACM and automatically builds static data structures, see Figure 46, to be used during the development process of application tiers, see Figure 47. The data structures convey to programmers a clear awareness about the FGACM to be dynamically implemented at runtime. The awareness is achieved while programmers write source code not at compilation time as many other research alternatives do. These assertions were verified and confirmed while source code for DbProof was being written.

### 5.3.4 Preservation of CLI Advantages

This sub-section evaluates how the third research question of the second level is answered by the DACA. The research question is: *“Is it possible to keep those advantages (of CLI) on the proposed solution to implement FGACM?”*. The use of CLI to build business tiers presents several advantages. The advantages may be classified in two major groups: the set of services provided by CLI and performance of CLI.

CLI are used at the DACC level only and there is no other component between CLI and RDBMS. Thus, the answer to the research question has to be found inside the DACC.

#### Preservation of CLI Services

The DACC provide a wide set of services. Some of them are geared to address the dynamicity of FGACM but others are geared to allow application tiers to execute CRUD expressions. The latter services are specified through DACC and are the services to be compared with those provided by CLI. As previously mentioned, DACC ensures the two of the access modes provided by CLI. The remaining access modes were not implemented but may easily be included in a future version of the DACC and the DACA. The remaining main CLI services are practically mapped one by one into the DACC: all scrolling services are available, different instantiations contexts of LMS are available, transactions are available, etc. Services such as access to metadata of

returned relations are not addressed because in the context of the DACA they are not relevant. If any other additional service is considered relevant it will certainly have an easy implementation in the DACC. The simplest approach is to wrap the intended service/method as shown in Figure 60. Figure 40, Figure 41 and Figure 42 show that relevant services for the execution of CRUD expressions are all available in the DACA.

```
ResultSetMetaData getMetadata() throws SQLException {  
    return rs.getMetadata();  
}
```

Figure 60. Wrapping approach to provide the *getMet*

### Preservation of CLI Performance

To prove that the performance advantage of CLI is kept, a performance assessment has been carried out. Results have shown that the DACC impact may be considered irrelevant for all scenarios but  $SS_r$ . Regarding  $SS_r$ , in the worst situation (2000 rows and no join), the % overhead is 7.0%, 3.6%, 2.5% and 2.0% for FoRo, FoUp, ScRo and ScUp, respectively. These results may be considered significant but a closer analysis shows that the % overhead has a deep dependency on many factors as it may be inferred from the graphics, namely on CRUD expressions complexity. A CRUD expression with a simple join led to a decay of about 50% in the % overhead. Thus, in real database applications, where most of the CRUD expressions are more complex than those herein used and tables are populated with thousands of rows, the % overhead will have an irrelevant impact on the overall performance.

When compared with other approaches, it is our opinion that the overhead induced by DACA is very probably lower than theirs, even for those approaches that use static enforcement mechanisms. The authors of these approaches argue that their solutions induce no overhead at all, just because policies are directly translated into CRUD expressions. This argument should only be used if they had compared their approaches with solutions with no access control, as it was done in this thesis. Only comparing with solutions with no access control it is possible to evaluate the impact of the access control on the overall performance. Moreover, the latency of the DACA is minimum because the decisions and the mechanisms are both located at the client application level.

## **5.4 Summary**

This chapter was focused on presenting the DACA proof of concept and it is organized in three sections: the presentation of the implemented platform, the performance assessment and, finally, the results evaluation.

The implemented platform is based on the DACA and includes a database application based on the Microsoft Northwind database. Some users and roles were defined and implemented. Additionally, roles are assigned and unassigned at runtime to convey a context of evolving FGACM. The Security Configurator was split in two different components to ease its development process. All DACA components were successfully implemented.

The performance assessment compared a solution based on the DACA and an equivalent solution but without any access control mechanisms. The collected results show that the induced overhead for the Direct Access Mode is marginal. Even for the Indirect Access Mode, only the read protocol induces measurable overhead values. The overheads are measurable because the running conditions were favorable to the existence of overheads. In real situations, where databases include thousands of rows and CRUD expressions are more complex, the induced overhead will become residual again.

The results evaluation proved that the DACA answered all the research questions of this thesis positively.

The next chapter presents the final conclusions.

## 6 Conclusion

This chapter is organized in four topics. First, a review about the work performed is presented. Second, the main contributions are highlighted. Third, a discussion about some adjacent aspects is taken and presented. Four, a perspective is presented for future work.

### 6.1 Overview

This thesis presents an architecture to enforce FGACP dynamically at the level of business tiers based on CLI, herein known as the DACA. The evolution from CLI towards DACA followed a three step approach: modelization of CLI, componentization of CLI and, finally, dynamic access control on CLI. The modelization of CLI lead to an effort to devise a model based on CLI to represent schemas of database objects [Pereira, '10b; Pereira, '11b]. The componentization of CLI lead to an effort to devise an architecture for components based on CLI [Pereira, '11a; Pereira, '11c; Pereira, '12b; Pereira, '13d; Pereira, '13e]. The dynamic access control on CLI leverages all previous work to devise the DACA [Pereira, '12d; Pereira, '12c; Pereira, '13d]. This three step approach was very important to successfully and separately overcome the distinct dimensions of CLI drawbacks. Without a model perspective and without a component perspective of CLI, the DACA, and mainly DACC, could hardly have been devised. The DACA comprises three main components: a server component where metadata about FGACM are maintained, a client component responsible for the implementation of dynamic FGACM and a proxy component placed between the server and the client components. Basically, programmers no longer have access to CLI but instead they have access to a component providing a similar set of services as CLI do. This component, DACC, is able to adapt itself dynamically, at runtime, when policies evolve over time. A proof of concept was designed and a performance assessment was carried out to evaluate the DACA against the research questions.

Finally, the DACA was evaluated against the research questions. The results show that the DACA answers to all the research questions positively: 1) Solutions based on the DACA are able to implement FGACM dynamically built and updated at runtime; 2) the DACA completely controls the CRUD expressions that users are authorized to issue this way preventing any security gap; 3) From the metadata of FGACM it is possible to build data structures to convey a complete awareness of FGACM to programmers of application tiers and, finally, 4) the DACC rely on and keep advantages of CLI.

## 6.2 Contributions

The outcome of this thesis is divided in one main contribution and four second level contributions. The main contribution is the DACA. The DACA is an architecture able to address and overcome some aspects still not addressed by current commercial and academic proposals. The main aspect is the implementation of dynamic FGACM on business tiers of relational database applications based on CLI. Solutions based on the DACA are able to implement FGACM dynamically, at runtime, and keep them updated even if policies evolve over time. Additionally, some other relevant aspects have also been addressed and overcome by the DACA such as the exploitation and preservation of CLI features, increased secure mechanisms and awareness of FGACM at development time of application tiers. Additionally, the DACA is the result of a continuous research that started on modelization of CLI, followed to the componentization of CLI and only then the DACA was devised and designed. Modelization and componentization of CLI are also two second level contributions of this thesis.

The remaining two second level contributions are also related to CLI. One is focused on a proposal to increase the performance of CLI whenever concurrency is needed on LMS, which is presented in the Annex B. Two approaches were designed. One based on standard JDBC [Pereira, '07b] and the other based on a new JDBC with embedded concurrent services [Gomes, '11]. The last contribution [Pereira, '13b], presented in the Annex C, is focused on an architecture to implement multi-propose components from CLI. Multi-propose components are able to address different organizational and different runtime needs .

## 6.3 Discussion

The DACA was evaluated against the research objectives initially defined. There are other issues that also deserve a brief reflection, in spite of not being key aspects of this thesis. As such, a brief description is presented about seven different aspects: FGACP, scalability of the DACA, maintainability of the DACA, autonomic computing and the DACA, configurability of the DACA, usability of the DACA and applicability of other technologies than CLI.

### FGACP

This work is about how to implement FGACM on components relying on CLI and not about FGACP. The DACA is independent from the policies to be applied. In practice, FGACP can be defined using some of the approaches presented by other authors. The only constraint is that from the established FGACP, the DACA requires Business Schemas as input and the associated CRUD expressions.

### Scalability

Unlike some approaches to implement access control mechanisms, such as those based on the centralized and mixed architectures, their implementation in the DACA is completely distributed. Each client application is responsible for two fundamental aspects: to decide upon granting or denying the access to protected data and to enforce the decision. There is no central system interfering in this process. It is completely distributed. Regarding the Policy Server, if for some reason, it exceeds an established threshold of loading, the Policy Server may be deployed using any common horizontal scalability approach.

### Maintainability

Business Logic and the correspondent FGACM are automatically built and updated at runtime. This feature clearly eliminates the need to carry out maintenance activities at the Business Logic and at the FGACM level. Moreover, any maintenance activity, at the level of the policies, is deployed and implemented automatically in all client applications. This is clearly different from what happens in all implementations of other authors as presented and described in subsection 2.4.2.

### Autonomic Computing

An autonomic system is characterized by making decisions on its own. It permanently checks the context and, based on policies, it automatically adapts itself. The DACA is not an autonomic system but systems based on the DACA are easily integrated in autonomic systems. An autonomic system prepared to detect situations where FGACP need to be dynamically adapted, can use the DACA to dynamically adapt the mechanisms.

### Configurability

In this thesis we presented an approach for a partial configuration process of FGACM metadata. The process is substantially automated if an enhanced tool similar to the one presented in [Pereira, '11b] is used. The new tool would create Business Schemas automatically from CRUD expressions and would also aggregate sibling CRUD schemas. Moreover, the tool could also automate the process to obtain the basic set of Business Schemas and CRUD expressions to access databases on a table basis as O/RM tools and LINQ do.

### Usability

CLI are very poor regarding their usability. The DACA overcomes some of the most relevant aspects of their lack of usability:

- Whenever CLI are being used, programmers need to master database schemas to deal with each retrieved attribute of each CRUD expression. With DACA, IRead, IInsert and IUpdate interfaces provide schema-driven getter and setter methods, avoiding the need to master database schemas for each CRUD expression.
- Whenever CLI are being used, programmers need to know the instantiation context of LMS. With the DACA, only the valid methods are available, this way avoiding runtime exceptions.
- Whenever CLI are being used, there is no easy way to link CRUD expressions and the applications they assist. With the DACA, the linkage is provided by schema-driven and type safe methods.
- The DACA, unlike CLI, transform runtime errors into compile errors. If the name of an attribute is modified, then the associated Business Schemas (IRead, IUpdate and IInsert interfaces) are also modified. Then, when the application tier is re-compiled, the compiler will detect all errors where the source-code of application tiers was not updated. With CLI, names of attributes are encoded inside strings, this way preventing any disconformity from being detected at compile time.

### Applicability

JDBC was the main API used to build DACC. While the Policy Server and the Policy Manager do not rely on any specific architecture, the DACC completely relies on the architecture of CLI. In

order to evaluate the possibility of using other tools than CLI to build the DACC, a successful attempt was achieved with ADO.NET. The implementation in ADO.NET was mainly done to evaluate if the main architectural aspects of the DACC are flexible enough to be used with different architectural paradigms. There were some technical aspects that needed some adjustments, but the final result is a DACC based on ADO.NET. The adjustments were mainly related with:

- scrolling policies on LMS – ADO.NET uses an index to choose the selected row.
- functionalities of LMS – scrollability and updatability concepts are restricted to scrollable and updatable.

The only difference between the DACC used on the proof of concept and the one based on ADO.NET is that it was assumed that the building process of Business Entities at runtime is also possible in the .NET framework. Thus, the ADO.NET version defines all the Business Schemas and CRUD expressions presented in Table 4 at development time. The behavior of the component is dynamically updated whenever a role is assigned or unassigned but the automatic building process of Business Entities is not implemented.

A DbProof based on ADO.NET is also available from here (url: ned.av.ia.it.pt; username: DACA; password: guest) and the main GUI, correspondent to the DbProof, is presented in Figure 61.

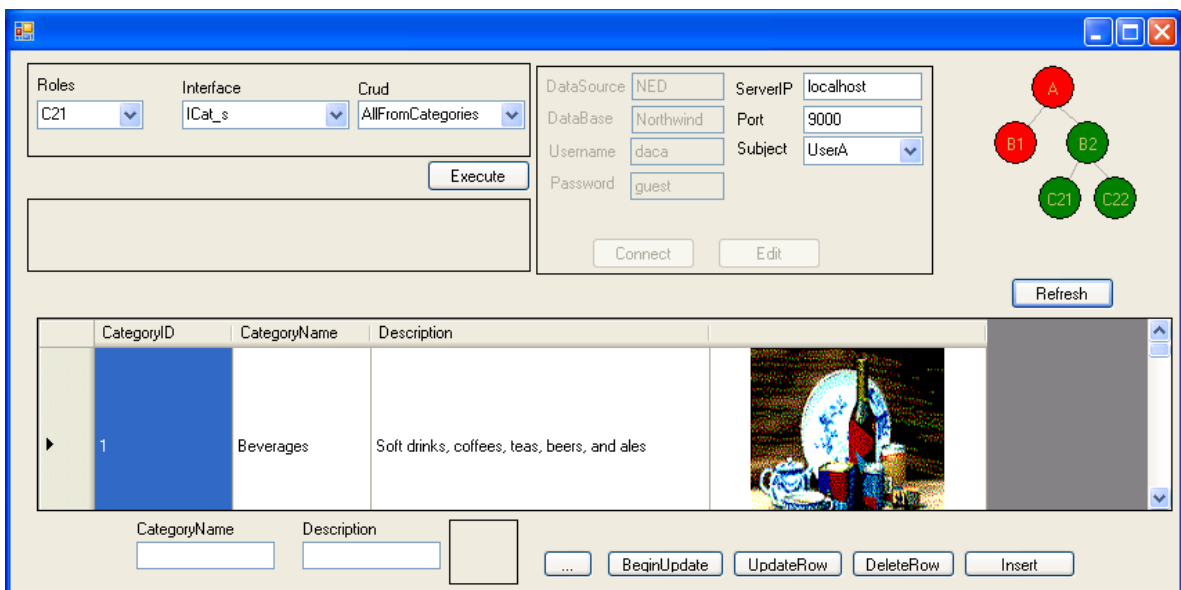


Figure 61. DbProof implemented in ADO.NET.

From my background, I foresee that DACC may rely on any CLI or even on any technologic paradigm used for building business tiers. Regarding O/RM tools, they should not be considered as an option because they are mostly oriented to handle database tables as entity classes which is too restrictive to most database applications. CRUD expressions may also be handled by O/RM tools but that is not their focus. Additionally, O/RM tools use CLI as the underlying middleware, this way behaving as an additional layer in the DACA. Regarding LINQ,



due to the static syntax validation of the SQL at writing time, it would be interesting to evaluate how to adapt DACA to rely on LKINQ instead on CLI.

## **6.4 Future Work**

Future work is organized around seven main objectives. The first one aims at extending the actual DACC to support additional access modes of CLI. The second one aims at devising a FGACP and the correspondent tool to validate accesses based on the access modes of CLI. The third one aims to deepen the research already started to provide a thread-safe implementation of the DACC. The fourth one aims at deepen the research already started to provide a multi-function propose for components based on CLI. The fifth one is focused on extending the FGACP to the runtime values that are used on CRUD expressions. The sixth is focused on devising a model to allow orchestration of Business Entities. The seventh and last one is focused on re-designing to be based on LINQ instead on CLI.

### **6.4.1 Extending DACC to Support Additional Access Modes**

The DACC supports the Direct Access Mode and the Indirect Access Mode of CLI. However, CLI support other additional access modes, such as access to stored procedures and execution of CRUD expression in batch mode. Thereby, extending the DACC to support the other access modes is considered an important step to cover all the access modes provided by CLI. The batch mode uses a buffer to store CRUD expressions. Whenever required, the stored CRUD expressions are processed as batch job. Stored procedures are software units stored in a RDBMS and available to client applications. Stored procedures may execute any task but are mainly used to access to stored data.

### **6.4.2 Fine-grained Access Control Policies for the DACA**

The presented work does not cover FGACP. To address this aspect, future work can be organized in two complementary steps: model definition and FGACP definition.

The model herein presented for the implementation of FGACM on CLI, shown in Figure 43, needs to be improved to be usable in real database applications. Beyond the model, an improved Security Configurator is needed to keep the configuration process as easy as possible.

Regarding the FGACP definition, DACA is not dependent on any FGACP, which is considered a key advantage. Regarding the Direct Access Mode, this independency potentially allows the use of any of the proposed approaches that have as output authorized CRUD expressions. Regarding the Indirect Access Mode, current approaches do not provide any solution. Eventually, the use of the Indirect Access Mode could also be inferred from the authorized FGACP. This is an open issue deserving a thoroughly research to devise a complete model for FGACP to be dynamically enforced on CLI.

### **6.4.3 Concurrent Approach of Call Level Interfaces**

CLI do not provide any mechanism to support concurrency. This may be considered another drawback of CLI if several threads need to access data using the same database connection. In order to evaluate the impact of implementing a thread-safe version of CLI, two researches were carried

out, both having promising results as outcome. The first research evaluated the implementation of thread-safe services at the level of CLI [Pereira, '10a]. The second research evaluated the implementation of thread-safe services at the JDBC driver level (TDS - Tabular Data Stream) [Gomes, '11] [Microsoft, '12], see Figure 10. Several threads share one LMS to execute read, update, insert and delete actions.

The collected results show that significant performance improvement is achieved for both approaches when compared with the traditional approach where each thread manages and interacts with its own database connection. Annex B describes the results that have been achieved for both approaches.

The research already undertaken needs to be continued to deepen and also to validate the collected results. Then, the knowledge gained from the research should be used to devise a thread-safe implementation of the DACC.

#### 6.4.4 Multi-function Components

In [Pereira, '13b] a new architecture is presented to address a new research challenge: business tiers components aimed at addressing different organizational and runtime needs. Organizational needs may include separation of roles for the development processes of business tiers and application tiers. Runtime needs may include the need to support new business requirements at runtime. Annex C describes the results that have been achieved for both approaches.

#### 6.4.5 Extending FGACP to the Runtime Values of CRUD expressions

The runtime values that are used on CRUD expressions are critical because they are dynamically defined by users at runtime, this way enabling users to request the access to different data in each execution cycle. We present three examples to justify our claims. The first one is based on a native Select expression, the second one is based on a native Update expression and, finally, the third one is based on modifying the contents of a record set containing data retrieved by a Select expression (in these cases the modifications are also committed to the host database). The following example is a simple Select expression.

```
Select t1.* from table1 t1, table2 t2
      Where t1.id = t2.t1_id and
            t1.value > pValue
```

The parameter (runtime value) *pValue* plays a key role to decide which data are retrieved from *table1*. In each individual execution cycle, the parameter may have a different value, this way retrieving a different set of records from *table1*. To overcome this source of possible security gaps, two approaches are used to implement the access control mechanisms: centralized approach and distributed approach. Regarding the centralized approach, the most common technique is the use of views (with [Rizvi, '04] or without query rewriting techniques). This technique conveys several drawbacks among which the lack of scalability is emphasized [Lopez, '02b; Valle, '02]. Regarding the distributed approach, two techniques were proposed: in [Chlipala, '10] is proposed a new predicate, identified as *known*, to model which information users already know, this way covering the points here under discussion but only superficially; in [Caires, '11] the policies are statically

enforced at the table columns level and not at the CRUD expressions level, leading to lack of flexibility.

The following example is the second example, which is a simple Update expression:

```
Update table1 t1
  set t1.value=pValue
  Where t1.id=pId
```

Similarly to the Select expression, this Update expression also uses parameters. The parameter *pValue* updates the attribute *value* of *table1* of a record identified by another parameter *pId*. Once again, parameters are user defined and play a key role on Update expressions to decide the data to be updated. The current techniques and their limitations, previously described for Select expressions, are also applied to Update expressions. The remaining types of CRUD expressions, Insert and Delete, convey similar limitations.

The last example is a very common situation on current tools that are used to develop business tiers, such as JDBC [Parsian, '05], Hibernate [Christian, '04], ADO.NET [Pablo, '07] and LINQ [Erik, '06]. The example shows that beyond the use of CRUD expressions, databases are also modifiable by executing protocols on data retrieved by Select expressions. The example shows that after retrieving data from a database, it is kept in record sets (*recordSet*) and then applications are allowed to update their content through an update protocol. In this case the attribute *attributeName* was updated to *value* and then the modification was committed. This case is different from the two previous ones because there is no evidence of any CRUD expression and users are modifying data they have been previously authorized to retrieve. Even so, we cannot despise the need to control the runtime values being used to modify the contents of those record sets and, therefore, used to modify the contents of databases. Beyond the update protocol, current tools also provide an insert protocol where users are also allowed to use runtime values.

```
recordSet=executeSelectExpression(sql)
recordSet.update("attributeName", value)
recordsSet.commit()
```

Currently, there isn't any known access control technique to enforce policies at the business tier level and able to statically control the provenance of runtime values that are used on actions issued against databases. To overcome this situation a first approach has been presented [Pereira, '13g] where parameters are statically driven by access control policies enforced at the business tier level.

### 6.4.6 Orchestration of Business Entities

The DACA does support orchestration of Business Entities, this way preventing FGACM to be implemented also at a higher level. For example, a role may comprise a specific task where the execution of CRUD expressions must follow a specific sequence.

### 6.4.7 The DACA Based on LINQ

As already mentioned, the DACA may be easily based on other tools beyond the CLI. This is true if the tool is not LINQ. LINQ is a C# language extension aimed at editing SQL statements whose syntax is statically validated at editing time. This powerful feature cannot be removed from LINQ and, thus,

it seems to be a significant challenge to simultaneously enforce FGACP while the SQL statements are being edited.

## References

- [Agrawal, '02] Agrawal, Rakesh, Kiernan, Jerry, Srikant, Ramakrishnan and Xu, Yirong: "Hippocratic databases." 28th Int. Conf. on Very Large Data Bases, VLDB Endowment, Hong Kong, China (2002), 143-154.
- [Andy, '08] Andy, Maule, Wolfgang, Emmerich and David, S. Rosenblum: "Impact analysis of database schema changes." 30th Int. Conf. on Software Engineering, ACM, Leipzig, Germany (2008), 451-460.
- [Anwar, '12] Anwar, Mohd and Fong, Philip W. L.: "A visualization tool for evaluating access control policies in facebook-style social network systems." 27th Annual ACM Symposium on Applied Computing, ACM, Trento, Italy (2012), 1443-1450.
- [Ao, '04] Ao, Xuhui and Minsky, Naftaly H.: "On the role of roles: from role-based to role-sensitive access control." Proceedings of the ninth ACM symposium on Access control models and technologies, ACM, Yorktown Heights, New York, USA (2004), 51-60.
- [Bachmann, '00] Bachmann, Felix, Bass, Len, Buhman, Charles, Comella-Dorda, Santiago, Long, fred, Robert, John E., Seacord, Robert C. and Wallnau, Kurt C. (2000). Volume II: Technical Concepts of Component-Based Software Engineering. <http://www.sei.cmu.edu/library/abstracts/reports/00tr008.cfm>, CMU/SEI.
- [Barker, '08] Barker, Steve: "Dynamic Meta-level Access Control in SQL." 22nd Annual IFIP WG 11.3 Working Conf. on Data and Applications Security, Springer-Verlag, London, UK (2008), 1-16.
- [Bauer, '07] Bauer, Christian and King, Gaving: "Java Persistence with Hibernate"; Manning, (2007).
- [Belokosztolszki, '03] Belokosztolszki, András, Eysers, David M., Pietzuch, Peter R., Bacon, Jean and Moody, Ken: "Role-based access control for publish/subscribe middleware architectures." 2nd Int. Workshop on Distributed Event-based Systems, ACM, San Diego, California (2003), 1-8.
- [Berners-Lee, '01] Berners-Lee, Tim, Hendler, James and Lassila, Ora (2001) "The Semantic Web." [Scientific American](#).
- [Bertino, '00] Bertino, Elisa, Castano, Silvana, Ferrari, Elena and Mesiti, Marco: "Specifying and enforcing access control policies for XML document sources." World Wide Web, 3, 3 (2000), 139-151.
- [Bhat, '03] Bhat, Viraj and Parashar, Manish (2003). A Middleware Substrate for Integrating Services on the Grid. [HiPC - High Performance Computing](#). T. Pinkston and V. Prasanna, Springer Berlin, Heidelberg. 2913: 373-382.
- [Bhatti, '05] Bhatti, Rafae, Bertino, Elisa and Ghafoor, Arif: "A Trust-Based Context-Aware Access Control Model for Web-Services." Distributed and Parallel Databases, 18, 1 (2005), 83-105.
- [Blair, '09] Blair, G., Bencomo, N. and France, R. B.: "Models@ run.time." Computer, 42, 10 (2009), 22-27.
- [Bond, '07] Bond, Rebecca, See, Kevin Yeung-Kuen, Wong, Carmen Ka Man and Chan, Yuk-Kuen Henry: "Understanding DB2 9 security", (2007).
- [Bonner, '97] Bonner, Anthony: "Transaction datalog: a compositional language for transaction programming." Intl Workshop on Database Programming Languages, Springer, LNCS(1997), 373-395.
- [Bracciali, '05] Bracciali, Andrea, Brogi, Antonio and Canal, Carlos: "A formal approach to component adaptation." Journal of Systems and Software, 74, 1 (2005), 45-54.

- [Buneman, '06] Buneman, Peter, Chapman, Adriane and Cheney, James: "Provenance management in curated databases." ACM SIGMOD Int. Conf. on Management of Data, ACM, Chicago, IL, USA (2006), 539-550.
- [Caires, '11] Caires, Luís, Pérez, Jorge A., Seco, João Costa, Vieira, Hugo Torres and Ferrão, Lúcio: "Type-based access control in data-centric systems." 20th European conference on Programming Languages and Systems: part of the joint European conferences on theory and practice of software, Springer-Verlag, Saarbrücken, Germany (2011), 136-155.
- [Carminati, '09a] Carminati, Barbara, Ferrari, Elena, Heatherly, Raymond, Kantarcioglu, Murat and Thuraisingham, Bhavani: "A semantic web based framework for social network access control." 14th ACM Symposium on Access Control Models and Technologies, ACM, Stresa, Italy (2009a), 177-186.
- [Carminati, '06] Carminati, Barbara, Ferrari, Elena and Perego, Andrea (2006). Rule-Based Access Control for Social Networks On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops. R. Meersman, Z. Tari and P. Herrero, Springer Berlin / Heidelberg. 4278: 1734-1744.
- [Carminati, '09b] Carminati, Barbara, Ferrari, Elena and Perego, Andrea: "Enforcing access control in Web-based social networks." ACM Trans. Inf. Syst. Secur., 13, 1 (2009b), 1-38.
- [Chaudhuri, '07] Chaudhuri, S., Dutta, T. and Sudarshan, S.: "Fine Grained Authorization Through Predicated Grants." IEEE 23rd ICDE - Int. Conf. on Data Engineering, Istanbul, Turkey (2007), 1174-1183.
- [Chlipala, '10] Chlipala, Adam: "Static checking of dynamically-varying security policies in database-backed applications." 9th USENIX Conf. on Operating Systems Design and Implementation, USENIX Association, Vancouver, BC, Canada (2010), 1-14.
- [Christian, '04] Christian, Bauer and Gavin, King: "Hibernate in Action"; Manning Publications Co., (2004).
- [Cook, '05] Cook, William and Ibrahim, Ali (2005) "Integrating programming languages and databases: what is the problem?"
- [Cooper, '07] Cooper, Ezra, Lindley, Sam, Wadler, Philip and Yallop, Jeremy: "Links: Web Programming Without Tiers." 5th Intl Conf on Formal Methods for Components and Objects, Springer-Verlag, Amsterdam, The Netherlands (2007), 266-296.
- [Corcoran, '09] Corcoran, Brian J., Swamy, Nikhil and Hicks, Michael: "Cross-tier, Label-based Security Enforcement for Web Applications." 35th SIGMOD Int. Conf. on Management of Data, ACM, Providence, Rhode Island, USA (2009), 269-282.
- [Costa, '07] Costa, Cristóbal, Pérez, Jennifer and Carsí, José (2007). Dynamic Adaptation of Aspect-Oriented Components. Component-Based Software Engineering. H. Schmidt, I. Crnkovic, G. Heineman and J. Stafford, Springer Berlin / Heidelberg. 4608: 49-65.
- [Damiani, '02] Damiani, Ernesto, Vimercati, Sabrina De Capitani di, Paraboschi, Stefano and Samarati, Pierangela: "A fine-grained access control system for XML documents." ACM Trans. Inf. Syst. Secur., 5, 2 (2002), 169-202.
- [David, '90] David, Maier (1990). Representing database programs as objects. Advances in Database Programming Languages. F. Bancilhon and P. Buneman. N.Y., ACM: 377-386.
- [Decker, '08] Decker, Michael: "Requirements for a location-based access control model." 6th Int. Conf. on Advances in Mobile Computing and Multimedia, ACM, Linz, Austria (2008), 346-349.
- [Denning, '76] Denning, Dorothy E.: "A lattice model of secure information flow." Commun. ACM, 19, 5 (1976), 236-243.
- [Dinkelaker, '11] Dinkelaker, Tom: "AO4SQL: Towards an Aspect-Oriented Extension for SQL." 8th Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'11), Zurich, Switzerland (2011), 1-5.
- [Dwork, '08] Dwork, Cynthia: "Differential Privacy: A Survey of Results." 5th Intl. Conf. on Theory and Applications of Models of Computation, Springer-Verlag, Xi'an, China (2008), 1-19.
- [Eclipse, '12] Eclipse. (2012). "Eclipse " Retrieved 2012 Jul, from <http://www.eclipse.org/>.
- [Eder, '96] Eder, Johann: "View Definitions with Parameters." 2nd Intl Workshop on Advances in Databases and Information Systems, Springer-Verlag(1996), 170-184.

- [Eisenberg, '99] Eisenberg, Andrew and Melton, Jim (1999). Part 1: SQL Routines using the Java (TM) Programming Language. American National Standard for Information for Technology Database Languages - SQLJ, International Committee for Information Technology.
- [Elizondo, '10] Elizondo, Perla Velasco and Lau, Kung-Kiu: "A Catalogue of Component Connectors to Support Development with Reuse." *Journal of Systems and Software*, 83, 7 (2010), 1165-1178.
- [Emilin Shyni, '10] Emilin Shyni, C. and Swamynathan, S.: "Purpose Based Access Control for Privacy Protection in Object Relational Database Systems." *Data Storage and Data Engineering (DSDE), 2010 International Conference on*, (2010), 90-94.
- [Erik, '06] Erik, Meijer, Brian, Beckman and Gavin, Bierman: "LINQ: Reconciling Object, Relations and XML in the .NET framework." *ACM SIGMOD Intl Conf on Management of Data*, ACM, Chicago,IL,USA (2006), 706-706.
- [Fabry, '06] Fabry, Johan and D'Hondt, Theo: "KALA: Kernel Aspect Language for Advanced Transactions." *Proceedings of the 2006 ACM Symposium on Applied Computing*, ACM, Dijon, France (2006), 1615-1620.
- [Ferraiolo, '01] Ferraiolo, David F., Sandhu, Ravi, Gavrila, Serban, Kuhn, D. Richard and Chandramouli, Ramaswamy: "Proposed NIST Standard for Role-based Access Control." *ACM Trans. Inf. Syst. Secur.*, 4, 3 (2001), 224-274.
- [Ferraiolo, '92] Ferraiolo, David, Kuhn, D. Richard and Chandramouli, Ramaswamy: "Role-based access control." *15th National Computer Security Conference*, Baltimore - Maryland - USA (1992), 554-563.
- [Fischer, '09] Fischer, Jeffrey, Marino, Daniel, Majumdar, Rupak and Millstein, Todd: "Fine-Grained Access Control with Object-Sensitive Roles." *23rd ECOOP - European Conference on Object-Oriented Programming*, Springer-Verlag, Italy (2009), 173-194.
- [Flower, '02] Flower, Martin: "Patterns of Enterprise Application Architecture"; Addison-Wesley, (2002).
- [Fundulaki, '04] Fundulaki, Irimi and Marx, Maarten: "Specifying access control policies for XML documents with XPath." *9th ACM Symposium on Access Control Models and Technologies*, ACM, Yorktown Heights, New York, USA (2004), 61-69.
- [Garcia-Morchon, '10] Garcia-Morchon, Oscar and Wehrle, Klaus: "Modular context-aware access control for medical sensor networks." *15th ACM Symposium on Access Control Models and Technologies*, ACM, Pittsburgh, Pennsylvania, USA (2010), 129-138.
- [Gary, '07] Gary, Wassermann, Carl, Gould, Zhendong, Su and Premkumar, Devanbu: "Static checking of dynamically generated queries in database applications." *ACM Transactions on Software Eng. Methodology*, 16, 4 (2007), 14:01-14:27.
- [Gomes, '11] Gomes, Diogo, Pereira, Óscar Mortágua and Santos, Wilson (2011). JDBC (Java DB connectivity) concorrente. MSc Dissertation, University of Aveiro.
- [Graham, '72] Graham, G. Scott and Denning, Peter J.: "Protection: Principles and Practice." *Spring Joint Computer Conference*, ACM, Atlantic City, New Jersey (1972), 417-429.
- [Gregor Kiczales, '97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes Videira, Jean-Marc Loingtier, Joh Irwin: "Aspect-Oriented Programming." *ECOOP*, Jyvaskyla,Finland (1997), 220-242.
- [Gregory, '05] Gregory, Buehrer, Bruce, W. Weide and Paolo, A. G. Sivilotti: "Using parse tree validation to prevent SQL injection attacks." *5th Intl. Workshop on Software Engineering and Middleware*, ACM, Lisbon, Portugal (2005), 106-113.
- [Harrison, '76] Harrison, Michael A., Ruzzo, Walter L. and Ullman, Jeffrey D.: "Protection in operating systems." *Commun. ACM*, 19, 8 (1976), 461-471.
- [He, '09] He, Daisy Daiqin, Compton, Michael, Taylor, Kerry and Yang, Jian: "Access control: what is required in business collaboration?"; *20th Australasian Conference on Australasian Database - Volume 92*, Australian Computer Society, Inc., Wellington, New Zealand (2009), 105-114.
- [Heineman, '01] Heineman, George T. and Councill, William T.: "Component-Based Software Engineering: Putting the Pieces Together"; Addison-Wesley, (2001).

- [Hicks, '10] Hicks, Boniface, Rueda, Sandra, King, Dave, Moyer, Thomas, Schiffman, Joshua, Sreenivasan, Yogesh, McDaniel, Patrick and Jaeger, Trent: "An architecture for enforcing end-to-end access control over web applications." 15th ACM symposium on Access Control Models and Technologies, ACM, Pittsburgh, Pennsylvania, USA (2010), 163-172.
- [Hildmann, '99] Hildmann, Thomas and Barholdt, Jorg: "Managing trust between collaborating companies using outsourced role based access control." 4th ACM workshop on Role-based Access Control, ACM, Fairfax, Virginia, United States (1999), 105-111.
- [Hu, '11] Hu, Yuh-Jong and Yang, Jiun-Jan: "A semantic privacy-preserving model for data sharing and integration." Proceedings of the International Conference on Web Intelligence, Mining and Semantics, ACM, Sogndal, Norway (2011), 1-12.
- [Hur, '11] Hur, Junbeom: "Fine-grained data access control for distributed sensor networks." *Wirel. Netw.*, 17, 5 (2011), 1235-1249.
- [IBM, '07] IBM (2007). Hippocratic Database (HDB) Technology Projects. [IBM Research](#).
- [ISO, '03] ISO. (2003). "ISO/IEC 9075-3:2003." Retrieved [2011 May, from [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=34134](http://www.iso.org/iso/catalogue_detail.htm?csnumber=34134).
- [Iwaihara, '05] Iwaihara, Mizuho, Chatvichienchai, Somchai, Anutariya, Chutiporn and Wuwongse, Vilas: "Relevancy based access control of versioned XML documents." 10th ACM symposium on Access Control Models and Technologies, ACM, Stockholm, Sweden (2005), 85-94.
- [Jayapandian, '08] Jayapandian, Magesh and Jagadish, H. V.: "Automated creation of a forms-based database query interface." *Int. Conf. on Very Large Database*, 1, 1 (2008), 695-709.
- [José, '09] José, Filho, Bringel and Martin, Hervé: "A generalized context-based access control model for pervasive environments." 2nd SIGSPATIAL ACM Int. Workshop on Security and Privacy in GIS and LBS, ACM, Seattle, Washington (2009), 12-21.
- [Keller, '97] Keller, Wolfgang: "Mapping Objects to Tables - A Pattern Language." European Conference on Pattern Languages of Programming Conference (EuroPLOP), Irsse, Germany (1997), 1-26.
- [Kephart, '03] Kephart, J. O. and Chess, D. M.: "The vision of autonomic computing." *Computer*, 36, 1 (2003), 41-50.
- [Kim, '09] Kim, Kyu Il, Choi, Won Gil, Lee, Eun Ju and Kim, Ung Mo: "RBAC-based access control for privacy protection in pervasive environments." 3rd Int. Conf. on Ubiquitous Information Management and Communication, ACM, Suwon, Korea (2009), 255-259.
- [Kim, '10] Kim, Kyu Il, Kim, Won Young, Ryu, Joon Suk, Ko, Hyuk Jin, Kim, Ung Mo and Kang, Woo Jun: "RBAC-based access control for privacy preserving in semantic web." Proceedings of the 4th International Conference on Ubiquitous Information Management and Communication, ACM, Suwon, Republic of Korea (2010), 1-5.
- [Kirchberg, '10] Kirchberg, M. and Link, S.: "Hippocratic Databases: Extending Current Transaction Processing Approaches to Satisfy the Limited Retention Principle." System Sciences (HICSS), 2010 43rd Hawaii International Conference on, (2010), 1-10.
- [Koshutanski, '03] Koshutanski, Hristo and Massacci, Fabio: "An access control framework for business processes for web services." ACM workshop on XML security, ACM, Fairfax, Virginia (2003), 15-24.
- [Kuhn, '10] Kuhn, D. Richard, Coyne, Edward J. and Weil, Timothy R.: "Adding Attributes to Role-Based Access Control." *Computer*, 43, 6 (2010), 79-81.
- [Kulkarni, '08] Kulkarni, Devdatta and Tripathi, Anand: "Context-aware role-based access control in pervasive computing systems." 13th ACM Symposium on Access Control Models and Technologies, ACM, Estes Park, CO, USA (2008), 113-122.
- [Kung-Kiu, '07] Kung-Kiu, Lau and Zheng, Wang: "Software Component Models." *IEEE Trans. on Soft. Eng.*, 33, 10 (2007), 709-724.
- [Laddad, '03] Laddad, Ramnivas: "AspectJ in Action: Practical Aspect-Oriented Programming"; Manning Publications, /Greenwich,CT,USA, (2003).



- [Lammel, '06] Lammel, Ralf and Meijer, Erik: "Mappings Make data Processing Go 'Round: An Inter-paradigmatic Mapping Tutorial." *Generative and Transformation Techniques in Soft. Eng.*, LNCS-Springer-Verlag, Braga, Portugal (2006), 169-218.
- [Lampson, '74] Lampson, Butler W.: "Protection." *SIGOPS Operating Systems Review*, 8, 1 (1974), 18-24.
- [Lawson, '12] Lawson, Curt and Zhu, Feng: "Sentential access control." 50th Annual Southeast Regional Conference, ACM, Tuscaloosa, Alabama (2012), 303-308.
- [LeFevre, '04] LeFevre, Kristen, Agrawal, Rakesh, Ercegovac, Vuk, Ramakrishnan, Raghu, Xu, Yirong and DeWitt, David: "Limiting disclosure in hippocratic databases." 30th Int. Conf. on Very Large Databases, VLDB Endowment, Toronto, Canada (2004), 108-119.
- [Li, '05] Li, Jiangtao, Li, Ninghui and Winsborough, William H.: "Automated trust negotiation using cryptographic credentials." 2th ACM Int. Conf. on Computer and Communications Security, ACM, Alexandria, VA, USA (2005), 46-57.
- [Liu, '10] Liu, Donggang: "Efficient and distributed access control for sensor networks." *Wirel. Netw.*, 16, 8 (2010), 2151-2167.
- [Lopez, '02a] Lopez, Javier, Mana, Antonio, Pimentel, Ernesto, Troya, José M. and Valle, Mariemma Inmaculada Yague e del: "Access Control Infrastructure for Digital Objects." *Proceedings of the 4th International Conference on Information and Communications Security*, Springer-Verlag(2002a), 399-410.
- [Lopez, '02b] Lopez, Javier, Mana, Antonio and Valle, Mariemma Inmaculada Yague del: "XML-Based Distributed Access Control System." *Proceedings of the Third International Conference on E-Commerce and Web Technologies*, Springer-Verlag(2002b), 203-213.
- [Luo, '04] Luo, Bo, Lee, Dongwon, Lee, Wang-Chien and Liu, Peng: "QFilter: fine-grained run-time XML access control via NFA-based query rewriting." 13th ACM Int. Conf. on Information and Knowledge Management, ACM, Washington, D.C., USA (2004), 543-552.
- [Mann, '01] Mann, V., Matossian, V., Muralidhar, R. and Parashar, M.: "DISCOVER: An environment for Web-based interaction and steering of high-performance scientific applications." *Concurrency and Computation: Practice and Experience*, 13, 8-9 (2001), 737-754.
- [Mann, '02] Mann, Vijay and Parashar, Manish: "Engineering an interoperable computational collaboratory on the Grid." *Concurrency and Computation: Practice and Experience*, 14, 13-15 (2002), 1569-1593.
- [McSherry, '10] McSherry, Frank: "Privacy Integrated Queries: An Extensible Platform for Privacy-preserving Data Analysis." *Commun. ACM*, 53, 9 (2010), 89-97.
- [Mead, '11] Mead, Ged and Boehm, Anne: "ADO.NET 4 Database Programming with C# 2010"; Mike Murach & Associates, Inc., /USA, (2011).
- [Mecella, '06] Mecella, Massimo, Ouzzani, Mourad, Paci, Federica and Bertino, Elisa: "Access control enforcement for conversation-based web services." 15th Int. Conf. on World Wide Web, ACM, Edinburgh, Scotland (2006), 257-266.
- [Microsoft, '92] Microsoft. (1992). "Microsoft Open Database Connectivity." Retrieved Jul, 2012, from [http://msdn.microsoft.com/en-us/library/ms710252\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms710252(VS.85).aspx).
- [Microsoft, '10] Microsoft. (2010). "Visual Studio 2010." Retrieved 2012 Jul, from <http://www.microsoft.com/visualstudio/en-us>.
- [Microsoft, '12] Microsoft. (2012). "[MS-TDS]: Tabular Data Stream Protocol Specification." Retrieved Jul, 2012, from [http://msdn.microsoft.com/en-us/library/dd304523\(v=prot.13\).aspx](http://msdn.microsoft.com/en-us/library/dd304523(v=prot.13).aspx).
- [Microsoft, '13] Microsoft. (2013). "RecordSet (ODBC)." Retrieved Jun, 2012, from <http://msdn.microsoft.com/en-us/library/5sbfs6f1.aspx>.
- [Moffett, '91] Moffett, Jonathan D. and Sloman, Morris S.: "Content-dependent access control." *SIGOPS Oper. Syst. Rev.*, 25, 2 (1991), 63-70.
- [Moore, '91] Moore, James W.: "The ANSI binding of SQL to ADA." *Ada Letters*, XI, 5 (1991), 47-61.
- [Morin, '10] Morin, Brice, Mouelhi, Tejedine, Fleurey, Franck, Traon, Yves Le, Barais, Olivier and Jézéquel, Jean-Marc: "Security-Driven Model-based Dynamic Adaptation." *IEEE/ACM Int. Conf. on Automated Software Engineering*, ACM, Antwerp, Belgium (2010), 205-214.

- [OASIS, '12] OASIS. (2012). "XACML - eXtensible Access Control Markup Language." Retrieved Feb, 2012, from [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=xacml](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml).
- [Olson, '09] Olson, Lars E., Gunter, Carl A., Cook, William R. and Winslett, Marianne: "Implementing Reflective Access Control in SQL." 23rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Springer-Verlag, Montreal, P.Q., Canada (2009), 17-32.
- [Olson, '08] Olson, Lars E., Gunter, Carl A. and Madhusudan, P.: "A formal framework for reflective database access control policies." 15th ACM Int. Conf. on Computer and Communications Security, ACM, Alexandria, Virginia, USA (2008), 289-298.
- [Oo, '07] Oo, May Phy and Naing, Thinn Thu: "Access Control System for Grid Security Infrastructure." IEEE/WIC/ACM Int. Conf. on Web Intelligence and Intelligent Agent Technology - Workshops, IEEE Computer Society(2007), 299-302.
- [Oracle] Oracle. "Oracle TopLink." Retrieved Oct, 2011, from <http://www.oracle.com/technetwork/middleware/toplink/overview/index.html>.
- [Oracle] Oracle. "Using Oracle Virtual Private Database to Control Data Access." Retrieved Mar, 2013, from [http://docs.oracle.com/cd/B28359\\_01/network.111/b28531/vpd.htm#CIHBAJGI](http://docs.oracle.com/cd/B28359_01/network.111/b28531/vpd.htm#CIHBAJGI).
- [Oracle, '12a] Oracle. (2012a). "Connection." Retrieved 2012 Jul, from <http://docs.oracle.com/javase/6/docs/api/java/sql/Connection.html>.
- [Oracle, '12b] Oracle. (2012b). "Interface PreparedStatement." Retrieved 2012 Jul, from <http://docs.oracle.com/javase/6/docs/api/java/sql/PreparedStatement.html>.
- [Oracle, '12c] Oracle. (2012c). "Interface Statement." Retrieved 2012 Jul, from <http://docs.oracle.com/javase/6/docs/api/java/sql/Statement.html>.
- [Oracle, '12d] Oracle. (2012d). "JDBCConnectionPool." Retrieved 2012 Jul, from [http://docs.oracle.com/cd/E13222\\_01/wls/docs81/config\\_xml/JDBCConnectionPool.html](http://docs.oracle.com/cd/E13222_01/wls/docs81/config_xml/JDBCConnectionPool.html).
- [Oracle, '12e] Oracle. (2012e). "NetBeans." Retrieved 2012 Jul, from <http://netbeans.org/>.
- [Oracle, '13] Oracle. (2013). "ResultSet." Retrieved Jul, 2012, from <http://docs.oracle.com/javase/6/docs/api/java/sql/ResultSet.html>.
- [Pablo, '07] Pablo, Castro, Sergey, Melnik and Atul, Adya: "ADO.NET entity framework: raising the level of abstraction in data programming." ACM SIGMOD International Conference on Management of Data, ACM, Beijing, China (2007), 1070-1072.
- [Paci, '11] Paci, Federica, Mecella, Massimo, Ouzzani, Mourad and Bertino, Elisa: "ACConv -- An Access Control Model for Conversational Web Services." ACM Trans. Web, 5, 3 (2011), 1-33.
- [Padma, '09] Padma, J., Silva, Y. N., Arshad, M. U. and Aref, W. G.: "Hippocratic PostgreSQL." ICDE '09. IEEE 25th Int. Conf. on Data Engineering, (2009), 1555-1558.
- [Pan, '06] Pan, Chi-Chun, Mitra, Prasenjit and Liu, Peng: "Semantic access control for information interoperation." Proceedings of the eleventh ACM symposium on Access control models and technologies, ACM, Lake Tahoe, California, USA (2006), 237-246.
- [Parsian, '05] Parsian, Mahmoud: "JDBC Recipes: A Problem-Solution Approach"; Apress, /NY, USA, (2005).
- [Pereira, '10a] Pereira, Oscar M, Aguiar, Rui L and Santos, Maribel Yasmina: "Assessment of a Enhanced ResultSet Component for Accessing Relational Databases." ICSTE-Int. Conf. on Software Technology and Engineering, Puerto Rico (2010a), V1:194-201.
- [Pereira, '10b] Pereira, Oscar M, Aguiar, Rui L and Santos, Maribel Yasmina: "CRUD-DOM: A Model for Bridging the Gap Between the Object-Oriented and the Relational Paradigms." ICSEA 2010 - Int. Conf. on Software Engineering and Applications, Nice, France (2010b), 114-122.
- [Pereira, '11a] Pereira, Oscar M., Aguiar, Rui L and Santos, Maribel Yasmina: "An Adaptable Business Component Based on Pre-defined Business Interfaces." 6th ENASE: Evaluation of Novel Approaches to Software Engineering, Beijing, China (2011a), 92-103.
- [Pereira, '11b] Pereira, Oscar M., Aguiar, Rui L and Santos, Maribel Yasmina: "CRUD-DOM: A Model for Bridging the Gap Between the Object-Oriented and the Relational Paradigms - an Enhanced Performance Assessment Based on a case Study." International Journal On Advances in Software, 4, 1&2 (2011b), 158-180.

- [Pereira, '11c] Pereira, Oscar M., Aguiar, Rui L. and Santos, Maribel Yasmina: "A Reusable Business Tier Component with a Single Wide Range Static Interface." ECSA: 5th European Conference on Software Architecture, Springer Verlag - LNCS, Essen, Germany (2011c), 216-219.
- [Pereira, '12a] Pereira, Óscar Mortágua, Aguiar, Rui L. and Figueiral, Diogo Jorge Rolo (2012a). Arquitetura Dinâmica de Controlo de Acesso. Msc Dissertation, University of Aveiro.
- [Pereira, '12b] Pereira, Óscar Mortágua, Aguiar, Rui L. and Santos, Maribel Yasmina: "ORCA: Architecture for Business Tier Components Driven by Dynamic Adaptation and Based on Call Level Interfaces." 38th Euromicro Conf. on Software Engineering and Advanced Applications, Cesme, Izmir, Turkey (2012b), 183-191.
- [Pereira, '13a] Pereira, Óscar Mortágua, Aguiar, Rui L. and Santos, Maribel Yasmina (2013a). ABC Architecture - A New Approach to Build Reusable and Adaptable Business Tier Components Based on Static Business Interfaces. Evaluation of Novel Approaches to Software Engineering. L. A. Maciaszek and K. Hang, Springer-Verlag, Communications in Computer and Information Science. 275: 114-129.
- [Pereira, '13b] Pereira, Óscar Mortágua, Aguiar, Rui L. and Santos, Maribel Yasmina: "ABTC: Multi-propose Adaptable Business Tier Components Based on Call Level Interfaces." JPRIT - Journal of Research and Practice in Information Technology, (2013b), (submitted).
- [Pereira, '13c] Pereira, Óscar Mortágua, Aguiar, Rui L. and Santos, Maribel Yasmina: "A Concurrent Tuple Set Architecture for Call Level Interfaces." ICIS - 12th IEEE/ACIS International Conference on Computer and Information Science, Springer - Computer and Information Science Niigata, Japan (2013c), 143-158.
- [Pereira, '13d] Pereira, Óscar Mortágua, Aguiar, Rui L. and Santos, Maribel Yasmina: "DACA: Distributed Dynamic Access Control Architecture Based on Call Level Interfaces." IET Information Security, (2013d), (submitted).
- [Pereira, '13e] Pereira, Óscar Mortágua, Aguiar, Rui L. and Santos, Maribel Yasmina: "Reusable Business Tier Architecture Driven by a Wide Typed Service." 12th IEEE/ACIS - International Conference on Computer and Information Science, Niigata, Japan (2013e), (accepted).
- [Pereira, '13f] Pereira, Óscar Mortágua, Aguiar, Rui L. and Santos, Maribel Yasmina: "Reusable Business Tier Architecture Driven by a Wide Typed Service." ICIS 2013 - 12th IEEE/ACIS International Conference on Computer and Information Science, Niigata, Japan (2013f), 135-141.
- [Pereira, '13g] Pereira, Óscar Mortágua, Aguiar, Rui L. and Santos, Maribel Yasmina: "Runtime Values Driven by Access Control Policies Statically Enforced at the Level of the Relational Business Tiers." SEKE 2013 - Intl. Conf. on Software Engineering and Knowledge Engineering, Boston, USA (2013g), (accepted).
- [Pereira, '12c] Pereira, Óscar Mortágua, Aguiar, Rui L. and Santos, Maribel Yasmina: " ACADA - Access Control-driven Architecture with Dynamic Adaptation." SEKE - 24th Intl. Conf. on Software Engineering and Knowledge Engineering, Knowledge Systems Institute Graduate School, San Francisco, CA, USA (2012c), 387-393.
- [Pereira, '12d] Pereira, Óscar Mortágua, Aguiar, Rui and Santos, Maribel: "BTA: Architecture for Reusable Business Tier Components with Access Control." ICCSA - 12th Int. Conf. on Computer Systems and Applications, Springer Berlin / Heidelberg, Salvador, Bahia, Brazil (2012d), 682-697.
- [Pereira, '05] Pereira, Óscar Mortágua, Pinto, Joaquim Sousa and Anjo, António Batel (2005). abcNet - Alfabetização na NET. MSc Dissertation, University of Aveiro.
- [Pereira, '06] Pereira, Óscar Narciso Mortágua and Pinto, Joaquim Manuel Henriques Sousa: "Maintainability Assessment of an Enhanced Object-Oriented Approach for Wrapping Stored Procedures." Int. Conf. on Databases and Applications, Innsbruck-Austria (2006), 26-31.
- [Pereira, '07a] Pereira, Óscar Narciso Mortágua and Pinto, Joaquim Manuel Henriques Sousa: "Performance Assessment of an Enhanced Object-Oriented Approach for Wrapping Stored Procedures." IEEE Eurocon 2007 - International IEEE Conference on Computer as a Tool, Warsaw-Poland (2007a), 473-477.

- [Pereira, '05a] Pereira, Óscar Narciso Mortágua and Pinto, Joaquim Sousa: "Wrapping Stored Procedures: an Enhanced Object Oriented Approach." IEEE EUROCON 2005-Int. Conf. on "Computer as a Tool", IEEE, Belgrade-Serbia and Montenegro (2005a), 740-743.
- [Pereira, '07b] Pereira, Óscar Narciso Mortágua and Pinto, Joaquim Sousa: "Performance Assessment of an Enhanced Object-Oriented Approach for Wrapping Stored Procedures." IEEE Eurocon - Int. Conf. on Computer as a Tool, Warsaw, Poland (2007b), 473-477.
- [Pereira, '05b] Pereira, Óscar Narciso Mortágua, Pinto, Joaquim Sousa and Anjo, António José Batel: "Object Oriented Platform to RDBMS Stored Procedure." IADIS- Int. Conf. on Applied Computing, Carvoeiro-Algarve-Portugal (2005b), 99-106.
- [Raje, '12] Raje, Satyajeet, Davuluri, Chowdary, Freitas, Michael, Ramnath, Rajiv and Ramanathan, Jay: "Using ontology-based methods for implementing role-based access control in cooperative systems." 27th Annual ACM Symposium on Applied Computing, ACM, Trento, Italy (2012), 763-764.
- [Ribeiro, '01] Ribeiro, Carlos, Zúquete, André, Ferreira, Paulo and Guedes, Paulo: "SPL: An Access Control Language for Security Policies with Complex Constraints." Network and Distributed System Security Symposium, San Diego, CA, USA (2001), 89-107.
- [Rizvi, '04] Rizvi, Shariq, Mendelzon, Alberto, Sudarshan, S. and Roy, Prasan: "Extending Query Rewriting Techniques for Fine-grained Access Control." ACM SIGMOD Int. Conf. on Management of Data, ACM, Paris, France (2004), 551-562.
- [Roichman, '07] Roichman, Alex and Gudes, Ehud: "Fine-grained access control to web databases." 12th ACM symposium on Access Control Models and Technologies, ACM, Sophia Antipolis, France (2007), 31-40.
- [Russell, '05] Russell, A. McClure and Ingolf, H. Kruger: "SQL DOM: compile time checking of dynamic SQL statements." 27th Int. Conf. on Software Engineering, ACM, St. Louis, MO, USA (2005), 88-96.
- [Samarati, '01a] Samarati, Pierangela and Vimercati, Sabrina De Capitani di: "Access Control: Policies, Models, and Mechanisms." 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, Springer-Verlag(2001a), 137-196.
- [Samarati, '01b] Samarati, Pierangela and Vimercati, Sabrina De Capitani di: "Access Control: Policies, Models, and Mechanisms." Foundations of Security Analysis and Design (LNCS), 2171, (2001b), 137-196.
- [Sandhu, '94] Sandhu, R. S. and Samarati, P.: "Access Control: Principle and Practice." Communications Magazine, IEEE, 32, 9 (1994), 40-48.
- [Sandhu, '00] Sandhu, Ravi, Ferraiolo, David and Kuhn, Richard: "The NIST Model for Role-based Access Control: Towards a Unified Standard." 5th ACM Workshop on Role-based Access Control, ACM, Berlin, Germany (2000), 47-63.
- [Sandhu, '93] Sandhu, Ravi S.: "Lattice-Based Access Control Models." Computer, 26, 11 (1993), 9-19.
- [Sandhu, '96] Sandhu, Ravi S., Coyne, Edward J., Feinstein, Hal L. and Youman, Charles E.: "Role-Based Access Control Models." Computer, 29, 2 (1996), 38-47.
- [Schmoelzer, '06] Schmoelzer, G., Teiniker, E., Kreiner, C. and Thonhauser, M.: "Model-typed Component Interfaces." Software Engineering and Advanced Applications, 2006. SEAA '06. 32nd EUROMICRO Conference on, (2006), 54-63.
- [Sharifi, '09] Sharifi, Mahdi, Movahednejad, Homa, Tabatabaei, Sayed Gholam Hassan and Ibrahim, Suhaimi: "An effective access control approach to support web service security." 11th Int. Conf. on Information Integration and Web-based Applications & Services, ACM, Kuala Lumpur, Malaysia (2009), 529-535.
- [Shi, '09] Shi, Jie, Zhu, Hong, Fu, Ge and Jiang, Tao: "On the Soundness Property for SQL Queries of Fine-grained Access Control in DBMSs." 8th IEEE/ACIS Intl. Conf. on Computer and Information Science, IEEE Computer Society(2009), 469-474.

- [Staddon, '08] Staddon, Jessica, Golle, Philippe, Gagn, Martin and Rasmussen, Paul: "A content-driven access control system." 7th Symposium on Identity and Trust on the Internet, ACM, Gaithersburg, Maryland (2008), 26-35.
- [Swamy, '08] Swamy, N., Corcoran, B. J. and Hicks, M.: "Fable: A Language for Enforcing User-defined Security Policies." IEEE Symposium on Security and Privacy, (2008), 369-383.
- [Systems, '73] Systems, Secretary's Advisory Committee on Automated Personal Data (1973). Records, computers and the Rights of Citizen: Report of the Secretary's Advisory Committee on Automated Personal Data Systems, U.S. Department of Health, Education, and Welfare.
- [Szyperky, '02] Szyperky, Clemens, Gruntz, Dominik and Murer, Stephan: "Component Software - Beyond Object-Oriented Programming"; Addison-Wesley/ACM Press, (2002).
- [Tolone, '05] Tolone, William, Ahn, Gail-Joon, Pai, Tanusree and Hong, Seng-Phil: "Access control in collaborative systems." ACM Comput. Surv., 37, 1 (2005), 29-41.
- [Tootoonchian, '08] Tootoonchian, Amin, Gollu, Kiran Kumar, Saroiu, Stefan, Ganjali, Yashar and Wolman, Alec: "Lockr: social access control for web 2.0." 1st Workshop on Online Social Networks, ACM, Seattle, WA, USA (2008), 43-48.
- [Vagts, '11] Vagts, Hauke, Krempel, Erik and Fischer, Yvonne: "Access controls for privacy protection in pervasive environments." 4th Int. Conf. on Pervasive Technologies Related to Assistive Environments, ACM, Heraklion, Crete, Greece (2011), 1-8.
- [Valle, '02] Valle, Mariemma Inmaculada Yague del, Mana, Antonio, Lopez, Javier, Pimentel, Ernesto and Troya, José M.: "Secure Content Distribution for Digital Libraries." Proceedings of the 5th International Conference on Asian Digital Libraries: Digital Libraries: People, Knowledge, and Technology, Springer-Verlag(2002), 483-494.
- [Vimercati, '08] Vimercati, S. De Capitani di, Foresti, S. and Samarati, P. (2008). Recent Advances in Access Control - Handbook of Database Security. M. Gertz and S. Jajodia, Springer US: 1-26.
- [Vohra, '07] Vohra, Deepak (2007). CRUD on Rails - Ruby on Rails for PHP and Java Developers, Springer Berlin Heidelberg: 71-106.
- [Vuran, '06] Vuran, Mehmet C. and Akyildiz, Ian F.: "Spatial correlation-based collaborative medium access control in wireless sensor networks." IEEE/ACM Trans. Netw., 14, 2 (2006), 316-329.
- [W3C, '02] W3C. (2002). "The Platform for Privacy Preferences 1.0 (P3P1.0) Specification." Retrieved Aug, 2012, from <http://www.w3.org/TR/P3P/>.
- [W3C, '03] W3C. (2003). "Enterprise Privacy Authorization Language (EPAL 1.2)." Retrieved Aug, 2012, from <http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/>.
- [Waldman, '12] Waldman, Steve. (2012). "c3p0 - JDBC3 Connection and Statement Pooling." Retrieved 2012 Jul, from <http://www.mchange.com/projects/c3p0/index.html>.
- [Wang, '06] Wang, Chengwei: "Dynamic Access Control Prediction for Ordered Service Sequence in Grid Environment." IEEE/WIC/ACM Int. Conf. on Web Intelligence, IEEE Computer Society(2006), 145-151.
- [Wang, '11] Wang, Haodong, Sheng, Bo, Tan, Chiu C. and Li, Qun: "Public-key based access control in sensor net." Wirel. Netw., 17, 5 (2011), 1217-1234.
- [Wang, '07] Wang, Qihua, Yu, Ting, Li, Ninghui, Lobo, Jorge, Bertino, Elisa, Irwin, Keith and Byun, Ji-Won: "On the correctness criteria of fine-grained access control in relational databases." 33rd Int. Conf. on Very Large Data Bases, VLDB Endowment, Vienna, Austria (2007), 555-566.
- [Warner, '07] Warner, Janice, Atluri, Vijayalakshmi, Mukkamala, Ravi and Vaidya, Jaideep: "Using semantics for automatic enforcement of access control policies among dynamic coalitions." Proceedings of the 12th ACM symposium on Access control models and technologies, ACM, Sophia Antipolis, France (2007), 235-244.
- [William, '05] William, R. Cook and Siddhartha, Rai: "Safe query objects: statically typed objects as remotely executable queries." 27th Int. Conf. on Software Engineering, ACM, St. Louis, MO, USA (2005), 97-106.
- [Wonohoesodo, '04] Wonohoesodo, R. and Tari, Z.: "A role based access control for Web services." IEEE Int. Conf. on Services Computing, (2004), 49-56.

- [Yang, '10] Yang, Daoqi: "Java Persistence with JPA"; Outskirts Press, (2010).
- [Yang, '12] Yang, Jean, Yessenov, Kuat and Solar-Lezama, Armando: "A language for automatically enforcing privacy policies." SIGPLAN Not., 47, 1 (2012), 85-96.
- [Ye, '04] Ye, Wei, Heidemann, John and Estrin, Deborah: "Medium access control with coordinated adaptive sleeping for wireless sensor networks." IEEE/ACM Trans. Netw., 12, 3 (2004), 493-506.
- [Yu, '06] Yu, Cong and Jagadish, H. V.: "Schema summarization." 32nd Intl Conf on Very large data bases, VLDB Endowment, Seoul, Korea (2006), 319-330.
- [Yu, '03] Yu, Ting, Winslett, Marianne and Seamons, Kent E.: "Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation." ACM Trans. Inf. Syst. Secur., 6, 1 (2003), 1-42.
- [Zhang, '12] Zhang, Danfeng, Arden, Owen, Vikram, K., Chong, Stephen and Myers, Andrew. (2012). "Jif: Java + information flow (3.3)." Retrieved Aug, 2012, from <http://www.cs.cornell.edu/jif/>.
- [Zhang, '03] Zhang, Guangsen and Parashar, Manish: "Dynamic Context-aware Access Control for Grid Applications." 4th Int. Workshop on Grid Computing, IEEE Computer Society(2003), 101-108.
- [Zhu, '12] Zhu, Yan, Hu, Hongxin, Ahn, Gail-Joon, Yu, Mengyang and Zhao, Hongjia: "Comparison-based encryption for fine-grained access control in clouds." 2nd ACM Conf. on Data and Application Security and Privacy, ACM, San Antonio, Texas, USA (2012), 105-116.

## Annex A – Logical model for metadata of FGACM

This annex presents the logical model that was used in the DACA proof of concept. This model derives from the conceptual model presented in Figure 43.

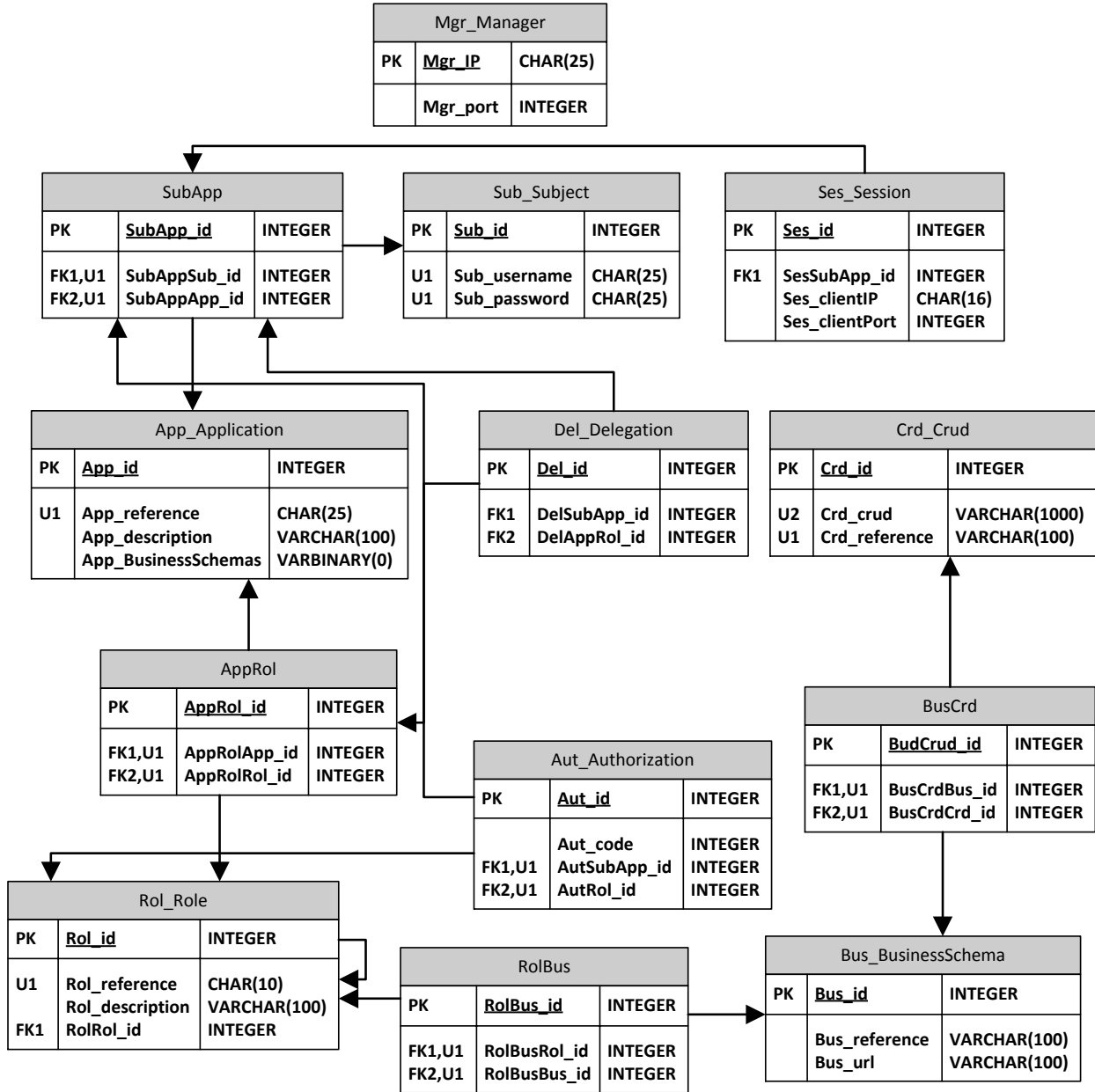


Figure 62. Logical model for *the proof of concept*.

Next follows a description of the main tables of the logical model: tables and attributes.

### Mgr Manager

Table Mgr\_Manager stores the required information to restore connections with Policy Managers whenever necessary.

Mgr\_IP – Policy Managers IP.

Mgr\_port –listening port.

#### Sub Subject

Table Sub\_Subject stores information to identify all the legitimate users.

Sub\_id – surrogate primary key.

Sub\_username – legitimate users' username.

Sub\_password – legitimate users' password.

#### App Application

Table App\_Application stores information about the legitimate applications.

App\_id – surrogate primary key.

App\_reference – application reference.

App\_description – application description.

App\_BusinessSchemas – file containing all the Business Schemas for this application. The value 0 (zero) should be replaced by MAX or any other adequate value able to hold the used business schemas.

#### Ses Session

Table Ses\_session stores information about subjects running applications and their associated Policy Manager.

Ses\_id – surrogate primary key.

Ses\_SubApp\_id – subject running an application.

Ses\_clientIP – associated Policy Manager IP.

Ses\_clientPort – associated Policy Manager port.

#### Rol Role

Table Rol\_Role stores definitions of roles. Roles are hierarchized.

Rol\_id – surrogate primary key.

Rol\_reference – role reference.

Rol\_description – role description.

RolRol\_id – parent Rol\_id.

#### Bus BusinessSchema

Table Bus\_BusinessSchema about Business Schemas.

Bus\_id – surrogate primary key.

Bus\_name – name to be used when data structures are built to convey awareness of FGACM.

Bus\_url – url for the Business Schema in App\_BusinessSchemas.



### Crd Crud

Table Crd\_Crud stores information about the supported CRUD expressions.

Crd id – surrogate primary key.

Crd reference – name to be used when data structures are built to convey awareness of FGACM.

Crd crud – CRUD expression.

### Aut Authorization

Table Aut\_Authorization stores the information about authorizations to subjects running applications to play roles.

Aut id – surrogate primary key.

Aut code – authorization codes. In this case there are two possibilities: authorization granted or denied.

AutSubApp id – assignment of applications to subjects.

AutRol id – assignment of roles to subjects running applications.

### Del Delegation

Table Del\_Delegation stores information about roles delegated to subjects running applications.

Del id – surrogate primary key.

DelSubApp id – assignment of applications to subjects.

DelAppRol – assignment of roles to subjects running applications.



## Annex B - Concurrency on CLI

Actions on LMS are tuple and protocol oriented and, while being executed, cannot be preempted to start another protocol. This restriction leads to several difficulties when applications need to deal with several tuples and several protocols at a time. The most paradigmatic case is the impossibility to cope with concurrent environments where several threads need to access to the same LMS instance, each one pointing to a different tuple and executing its own protocol. In order to evaluate the possibility of implementing a thread safe version of CLI, two approaches were followed: wrapper approach and the embedded approach.

### Wrapper approach

The wrapper approach uses standard CLI with native RDBMS protocols. Basically, the wrapper approach wraps LMS and exposes a set of services which are protocol-oriented to provide a thread-safe access to LMS. A paper has been published with the preliminary results [Pereira, '10a]. Some additional research has been done and published [Pereira, '13c]. The final results are herein presented in this annex.

### Embedded approach

The embedded approach is a more ambitious approach than the wrapper approach. The embedded approach uses modified RDBMS protocols, in our case Tabular Data Stream (TDS) [Microsoft, '12], to support concurrency on LMS. The work done in [Gomes, '11] showed that significant improvements are achieved if concurrency is applied directly on internal LMS data structures, even when compared with those obtained in the wrapper approach.

## B.1 CTSA- The Wrapper Approach

To deepen the research initiated in [Pereira, '10a] some additional work was done. The methodology to implement concurrency was tuned and a much more detailed performance assessment has been carried out. The outcome is a Concurrent Tuple Set Architecture (CTSA) to manage concurrency on LMS.

### B.1.1 CTSA Presentation

CTSA wraps CLI and provides thread safe services to access LMS. CTSA takes LMS as the main input entity hides its methods and exposes a new interface, which is thread-safe and, above all, is designed to improve concurrency between concurrent threads. In order to characterize LMS, their protocols may be organized in two orthogonal groups:

#### Scrollability

Scrollability defines the policies to scroll on the returned tuples. There are two mutual-exclusive possibilities: forward-only (move one tuple forward at a time) and scrollable (move

backward or forward any number of tuples).

Updatability

Updatability defines the policies to interact with the tuples kept inside LMS. There are two mutual-exclusive possibilities: read-only (only read protocol is supported) and updatable (read, insert, update and delete protocols are supported).

These different types of functionalities, scrollability and updatability, raise an important question: is it necessary to provide concurrency for all types of LMS? Regarding scrollability, forward-only LMS are very restrictive because they would oblige all threads to simultaneously point to the same tuple which could hardly happen in real scenarios. Regarding updatability, concurrency makes sense for both types: read-only and updatable. Read-only LMS always provide a subset of the functionalities of updatable LMS and, hence, in order to address and assess the most general case, we chose to implement a concurrent version for scrollable and updatable LMS. CTSA introduces the concept of *execution context* as the information needed to characterize, at any time, the interaction between a thread and a component based on the CTSA. The execution context of each thread comprises the protocol that is being executed and the current selected tuple. This concept is very important because it is the basis for the concurrent implementation of LMS. In concurrent environments, each thread must have a complete control on the tuple and on the protocol it is executing. If this is not ensured, a running thread may be preempted by another thread that changes the execution context. The first thread will never be aware about this situation and when it becomes the running thread it will execute its protocol in a different execution context. In order to keep full control on the execution context, each thread needs to access the LMS in exclusive mode and also to be able to assure that it runs on its own execution context. The former condition ensures that other threads are not allowed to change the execution context of protocols that are being executed. The latter condition ensures that at the beginning of any protocol, if necessary, every thread is able to restore its execution context. To decide upon which strategy to follow to implement both conditions, several possibilities were considered and tested. They may be classified in two distinct groups:

- method oriented: execution context is managed method by method;
- protocol oriented: execution context is managed at the protocol level.

Table 9 briefly shows the logic associated to each approach. To evaluate the implication of each approach, an assessment has been carried out. Results have shown that for the same scenarios,

<b>Method oriented</b>	<b>Protocol oriented</b>
1. get exclusive access 2. set execution context 3. execute method 4. store execution context 5. release exclusive access	1. get exclusive access 2. set execution context 3. while protocol is not over execute method 4. store execution context 5. release exclusive access

Table 9. Exclusive access mode approaches.

performance and concurrency improvement depend on the same variable but in opposite ways. They depend on the number of times that threads are preempted by other threads. Every time this occurs, a change in the execution contexts must be performed. When this number increases performance tends to decrease and concurrency tends to increase. When this number decreases, performance tends to increase and concurrency tends to decrease. Taking this into account, it would not be possible to carry out a detailed assessment for all scenarios. Thus, a structure was defined that would typify real scenarios:

```

while there is next tuple { // method oriented action
    execute a general block of code and/or a method oriented
    execute a protocol oriented action
    execute a general block of code and/or a method oriented action
    execute a protocol oriented action
    ...
}
    
```

In this structure the scrolling process takes place at an outer level, which is the most common practice, and it is performed by a single scrolling method. In the inner level, programmers are encouraged to avoid the execution of general code and/or method oriented actions while any protocol oriented action is being performed. General code inside protocol oriented actions extends the locking period, this way increasing the possibility to occur a request for a switching in the execution context. This request would not succeed and the thread would have to wait for the protocol to end. This leads to two unwanted situations: decay in performance and decay in concurrency, derived by the unsuccessful switching in the execution context only. Thus, the presented structure leads to the following options for the access mode:

- Implemented as method oriented for all actions that could be completely and undoubtedly accomplish with a single method;
- Implemented as protocol oriented for the remaining actions. Thus, access mode for Delete and Scroll protocols were considered as method oriented and access mode for Read, Insert and Update protocols were considered as protocol oriented.

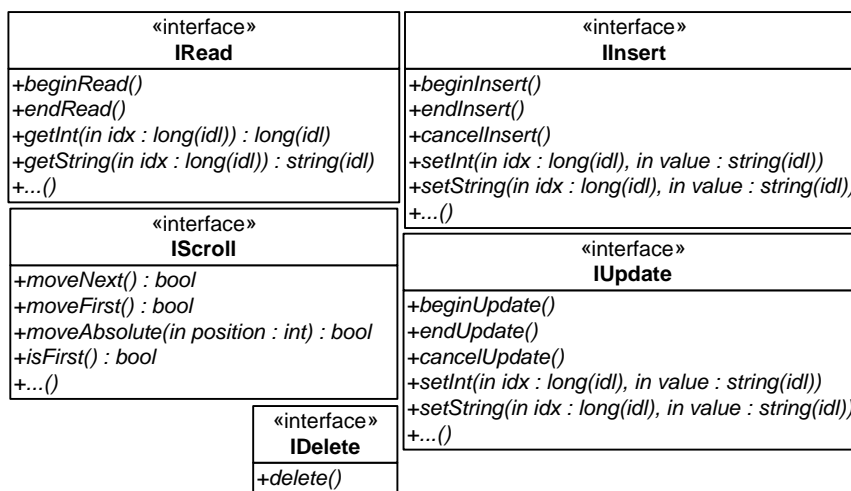


Figure 63. CTSA main protocols.

Figure 63 presents the interfaces for the five main protocols: IRead, IInsert, IUpdate, IDelete and IScroll. Only the main methods of IRead, IUpdate, IInsert and IScroll protocols have been presented in order to not overcrowd the class diagrams. Exclusive access mode based on the protocol oriented strategy needs a start event (beginRead, beginUpdate and beginInsert) to start the protocol and an end event (endRead, endUpdate and endInsert) to end the protocol. Exclusive access mode based on the method oriented strategy does not need any additional event but the methods themselves.

Figure 64 presents a simplified CTSA class diagram for a scrollable and updatable LMS. Each thread receives a new instance of a component based on the CTSA where all instances share the same LMS instance. *lms* is for the LMS instance, *currentTuple* is the index of the current selected tuple, *protocol* is the protocol being executed (if any) and *lock* is the object being used to grant the exclusive access mode to the LMS. *setExecutionContext* restores the execution context for the access mode just started and *storeExecutionContext* saves the current execution context just ended.

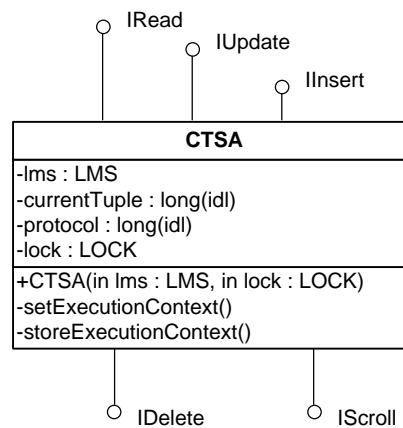


Figure 64. CTSA class diagram.

### B.1.3 Proof of Concept

This section evaluates CTSA using a proof of concept implemented in Java and JDBC. The ResultSet interface of JDBC API is used as a representative of LMS.

Figure 65 presents the CTSA constructor, its arguments and the initialization steps: *rs* is the LMS and *lock* is the object used to grant the exclusive access mode. After its instantiation, the execution context points to the tuple immediately before the first one, as happens with CLI (LMS use before first and after last tuple positions. Immediately after the execution of a Select statement, by default, the current selected tuple is the tuple before the first one).

```

CTSA(ResultSet rs, ReentrantLock lock) {
    this.rs = rs;
    this.lock = lock;
    currentTuple = beforeFirst;
}
  
```

Figure 65. CTSA constructor.

Figure 66 partially presents the read protocol. *beginRead* gets the exclusive access mode for the read protocol and then sets the execution context for the active thread. From now on, the thread may read the attributes of its current selected tuple. In Figure 66 it is only shown one method (*getInt*) to read attributes of type *Integer*. The *IRead* interface comprises all other necessary methods to support the additional data types. The protocol ends after the execution of *endRead* method which saves the current execution context and then releases the exclusive access mode. From now on, other protocols may be executed. The update and insert protocols, of which exclusive access mode is also protocol oriented, may be easily inferred from the read protocol. Thus their code will not be shown.

```
public void beginRead() throws SQLException {
    lock.lock();
    setExecutionContext();
}
public void endRead() throws SQLException {
    storeExecutionContext();
    lock.unlock();
}
public int getInt(int idx) throws SQLException {
    return rs.getInt(idx);
}
```

Figure 66. Partial view of *IRead* protocol.

Figure 67 shows the method *moveNext* which belongs to the scrolling protocol. Exclusive access mode of Scroll protocol is method oriented and, thus, all methods execute a lock and an unlock process. It is also necessary to set the execution context before moving the cursor one tuple forward to ensure the correct positioning. After moving the cursor, the execution context must also be saved. Method *next()* returns a *boolean*, indicating if in the current position it is or it is not After Last tuple.

```
public boolean moveNext() throws SQLException {
    lock.lock();
    setExecutionContext();
    try {
        bool = rs.next();
    } finally {
        lock.unlock();
    }
    storeExecutionContext();
    return bool;
}
```

Figure 67. Partial view of *IScroll* protocol.

Figure 68 presents the process used to set and to store the execution context. The concepts *Before First* tuple and *After Last* tuple are used to define the position immediately before and immediately after the first and the last tuple, respectively. These concepts are common in CLI and

need a special treatment regarding the execution context. In JDBC the first tuple is indexed by 1. In *storeExecutionContext*, *isBeforeFirst* (equal to -1) and *isAfterLast* (equal to -2) say if the cursor is pointing to the position immediately before or after the first or last tuple, respectively.

Figure 69 shows CTSA usage from users' perspective. The thread receives a CTSA instance in the constructor and accesses its LMS through the provided protocols. *moveNext* belongs to the scrolling protocol and therefore the exclusive access is method oriented. Before reading any attribute, it is necessary to get the exclusive access mode and set the execution context, which is achieved through the method *beginRead*. After reading all the attributes, the exclusive access is released through the method *endRead*.

```
private void setExecutionContext() throws SQLException {
    if ( currentTuple > 0 )
        rs.absolute( currentTuple );
    else if (currentTuple == beforeFirst)
        rs.beforeFirst();
    else rs.afterLast();
}
private void storeExecutionContext() throws SQLException {
    if ( rs.isBeforeFirst() )
        currentTuple = beforeFirst;
    else if ( rs.isAfterLast() )
        currentTuple = afterLast;
    else currentTuple = rs.getRow();
}
```

Figure 68. Set and store the execution context.

```
User( CTSA ctsa ) {
    this.ctsa = ctsa;
}
@Override
public void run() {
    // ... code
    try {
        while ( ctsa.moveNext() ) {
            ctsa.beginRead();
            id = ctsa.getInt( 1 );
            // ... read other attributes
            ctsa.endRead();
            // other actions
        }
    } catch ( SQLException ex ) {}
}
```

Figure 69. CTSA from users's perspective.

## B.1.4 CTSA Performance Assessment

Performance assessment was carried out comparing two entities known as the Component CTSA



(C-CTSA) and the Concurrent JDBC (C-JDBC). C-CTSA is responsible for evaluating components relying on the CTSA architecture and it is based on a component derived from the proof of concept here presented. C-JDBC is responsible for evaluating a concurrent approach based only on the standard JDBC API. The evaluation of both entities comprises a single façade: performance. The impossibility to assess all scenarios, led to a survey to define some scenarios that could be representative of common situations. To this end, we needed to identify the relevant aspects directly related and controlled by users of CTSA that could influence CTSA performance. Based both on empirical experiences and knowledge about CTSA, the aspects considered relevant were: the protocol being executed, the number of rows to be processed and the number of simultaneous running threads. Thus, three scenarios were defined for the three main types of protocols for both components: Select (s), Update (u) and Insert (i). Each scenario comprises a set of several numbers of tuples to be processed  $[nr]$  and a set of several numbers of simultaneous running threads  $[nt]$ . In order to formalize the entities' representation we define  $E_{(\alpha,p,\gamma)}([nt], [nr])$  where  $\alpha \in \{c-ctsa, c-jdbc\}$ ,  $p$  is for performance façade and  $\gamma \in \{s, u, i\}$ . To simplify,  $E_{(\alpha,p,\gamma)}([nt], [nr])$  is represented by default as  $E_{(\alpha,p,\gamma)}$ . To get a threshold for the performance of each entity, it was decided to create a favorable environment to C-JDBC and an unfavorable environment to C-CTSA to execute the scenarios. This way, the minimum performance of real scenarios should be delimited by the collected measurements. This issue will be addressed in more detail mainly after explaining the SQL Server behavior about LMS.

The test-bed comprises two computers: PC1 - Dell Latitude E5500, Intel Duo Core P8600 @2.40GHz, 4.00 GB RAM, Windows Vista Enterprise Service Pack 2 (32bits), Java SE 6, JDBC(sqljdbc4); PC2 - Asus-P5K-VM, Intel Duo Core E6550 @2,33 GHz, 4.00 GB RAM, Windows XP Professional Service Pack 3, SQL Server 2008. C-JDBC and C-CTSA are executed on PC1 and SQL Server runs on PC2. In order to promote an ideal environment the following actions were taken: the running threads were given the highest priority and all non-essential processes/services were cancelled in both PCs; a direct and dedicated network cable connecting PC1 and PC2 has been used in exclusive mode and performing 100MBits of bandwidth. Transactions were not used and *auto-Commit* has been always enabled (changes to LMS are automatically committed to the host database when protocols are ended). A new database was created in conformance with the schema presented in Figure 70 to assess both entities.

Std_Student	
Column Name	Data Type
Std_id	int
Std_firstName	varchar(25)
Std_lastName	varchar(25)
StdCrs_id	int
Std_regYear	smallint
Std_applGrade	float

Figure 70. Std\_Student schema.

In order to avoid some overhead added by SQL Server, some default SQL Server database properties were changed as, Auto Update Statistics = false and Recovery Model = Simple. Some important aspects are out of the scope of this study. Aspects as database server performance, network delays and memory consumption are not individually addressed but considered as part of the overall environment. This has been assumed because both entities share the same infrastructure. It is essential to recall SQL Server behavior, which is similar to most of other relevant relational database management systems, to completely understand the collected measurements of each scenario. When a Select statement is executed using a scrollable or an updatable LMS, SQL Server creates a server cursor with all the selected tuples. These tuples are dynamically transferred in blocks, from the server, to LMS whenever necessary. This means that at any time LMS may not have all the tuples but only a sub-set of all tuples. When users point to a tuple that is not present in the LMS, the Tabular Data Stream (TDS) [Microsoft] protocol discards the current LMS's content and fetches the block containing the desired tuple. This has a deep implication. If threads are always requesting tuples that are not present in the LMS, SQL Server has to transfer the correspondent block for each requested tuple. In an extreme scenario, each individual action on an LMS may imply the transference of a new block of tuples. From the previous statements, it is expected that the number of blocks to be transferred will increase when the number of tuples increases and also when the probability of a thread to request tuples that are not present in an LMS increases. Thus, to create different environments for both entities, the following decisions were taken:

C-JDBC (favorable environment): each thread has its own LMS and will always access tuples sequentially from the first one till the last one.

C-CTSA (unfavorable environment): three conditions were implemented: 1) all threads share the same LMS; 2) after accessing a tuple, each thread will give the opportunity to other threads to become the running thread by voluntarily leaving the running state - this will maximize the number of changes in the execution context; 3) each thread will have its own set of tuples, not shared with any other thread - this will maximize the number of blocks of tuples to be transferred from server cursors to LMS.

Table 10 shows the algorithm for the assessment of  $E_{(c-ctsa,p,\gamma)}$ . The same ResultSet is shared by all  $[nr]$  threads. Each thread executes its scenario for a group  $\psi=[nr]$  adjacent tuples and auto-suspends itself after accessing each tuple. The intersection of all  $\psi=\emptyset$ .

Table 11 shows the algorithm for the assessment of  $E_{(c-jdbc,p,\gamma)}$ . Each thread creates its own ResultSet (LMS) containing/inserting a group of  $\psi=[nr]$  adjacent tuples. The intersection of all  $\psi=\emptyset$ .

- |   |
|---|
| <ol style="list-style-type: none"> <li>1. Delete all rows from Std_Student</li> <li>2. Fill Std_Student with <math>[nr]*[nr]</math> rows (zero rows for insert)</li> <li>3. Start counter</li> <li>4. Select all rows from Std_Student into one single ResultSet</li> <li>5. Create all threads.</li> <li>6. Each thread (<math>\psi</math> tuples)             <ol style="list-style-type: none"> <li>6.1 for each tuple                 <ol style="list-style-type: none"> <li>6.1.1 read/update/insert (tuple)</li> <li>6.1.2 suspend thread</li> </ol> </li> <li>6.2 dies</li> </ol> </li> <li>7. Wait all threads to die</li> <li>8. Stop counter</li> </ol> |
|---|

Table 10. Algorithm for  $E_{(c-jdbc,p,\gamma)}$  assessment.

1. Delete all rows from Std\_Student
2. Fill Std\_Student with  $[nr]*[nt]$  rows (zero rows for insert)
3. Start counter
4. Create all threads.
5. Each thread:
  - 5.1 select  $\psi$  tuples into its own ResultSet
  - 5.2 for each tuple
    - 5.2.1 read/update/insert a tuple
  - 5.3 dies
6. Wait all threads to die
7. Stop counter

Table 11. Algorithm for  $E_{(c-jdbc,p,\gamma)}$  assessment

To contextualize the performance assessment environment some initial measurements were carried out to delimit the range of  $[nt]$  and  $[nr]$  to be used. In order to emphasize concurrency mechanisms, priority was given to the range of  $[nt]$  in detriment of  $[nr]$ . Values for these metrics were collected by empirical experimentation based on an iterative process. The idea is to gather a set of values for  $[nt]$  and  $[nr]$  that may be used to assess and compare the performance of both  $E_{(\alpha,p,\gamma)}$  entities. To accomplish this, both entities,  $E_{(c-ctsa,p,\gamma)}$  and  $E_{(c-jdbc,p,\gamma)}$  were executed under several combinations of  $[nt]$  and  $[nr]$  until the collected measurements comprise a range of behaviors considered satisfactory to accurately assess and compare the performance of both entities. After several iterations it was decided that the execution environment should be defined as:

$$[nt]=\{1,5,10,25,50,75,100,150,200,250,350,500\}$$

$$[nr]=\{5,10,25,50,75,100\}$$

In accordance with the requirements, this execution environment evaluates the performance by maximizing the number of simultaneous running threads in detriment of the number of tuples. With 500 threads and 100 tuples it was possible to accurately assess and foresee the performance behavior of both entities. This was the main reason for their acceptance. The intermediate collected measurements showed to be enough to obtain well defined charts for the behaviors of both entities. Just as a final note, some scenarios took some minutes to setup and to process the highest values of  $[nt]$  and  $[nr]$ . This knowledge was also considered to delimit the two top values ( $nr=100$  and  $nt=500$ ), this way avoiding any risk to successfully accomplish the collecting process of all necessary measurements. For the assessment, 100 raw measurements were collected for each  $E_{(\alpha,p,\gamma)}([nt],[nr])$  leading to  $(2 \times 3 \times 12 \times 6) \times 100 = 43,200$  raw measurements. Intermediate measurements were computed from the average of the 5 best measurements of each  $E_{(\alpha,p,\gamma)}([nt],[nr])$  leading to a total of  $2 \times 3 \times 12 \times 6 = 432$  measurements. The final measurements used in the next charts represent the ratios between  $E_{(c-jdbc,p,\gamma)}$  and  $E_{(c-ctsa,p,\gamma)}$  for each  $([nt],[nr])$ . In all charts the vertical axis is for the ratios and the horizontal axis is for the  $[nt]$ .

### **Select scenario**

The chart for the select scenario is shown in Figure 71. From it, it is clear that the ratios decrease whenever the number of tuples increases and whenever the number of threads increases (for most  $nt$ ). This derives from the fact that  $E_{(c-jdbc,p,\gamma)}$  have  $[nt]$  server cursors and each thread sequentially reads its own tuples from the first one till the last one. Thus, the transference of block of tuples only happens when a thread tries to read the next tuple that is after the last one contained in its own ResultSet. The probability for this to happen increases with the number of rows. Regarding  $E_{(c-ctsa,p,\gamma)}$

there is only one server cursor shared by all threads. The implemented Read scenario significantly increases the possibility of each thread to be requesting a tuple that is not present in the ResultSet and, therefore, to trigger a new transference of block of tuples. With other different strategies, for example where threads read shared sets of tuples, the block transference rate should be much lower leading to an increase on its performance. Another relevant issue is that the Select scenario is a light scenario mainly because the Select statement and Read protocol are very efficient when compared with the other protocols. Thus, the overhead induced by the blocks transference have a deeper impact in the overall performance. The impact increases with the number of tuples and the number of threads. Figure 72 presents a detailed view of all results. The ratio is greater or equal to 1 in 35 situations and less than 1 in 37 situations. It also shows that the highest ratio is 3.44 ( $nr=5, nt=10$ ) and the lowest one is 0.8 ( $nr=25, nt=100$ ). These results show that despite the unfavorable conditions for C-CTSA, it still achieves significant results. For example, the relative highest gain in performance (3.44) is much more significant than the relative highest lost in performance (0.8) and the average value is 1.21.

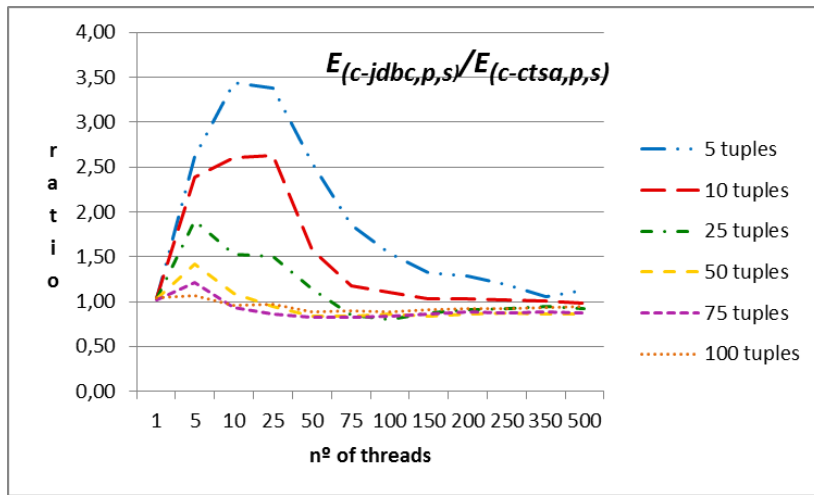


Figure 71.  $E_{(c-jdbc,p,s)} / E_{(c-ctsa,p,s)}$  chart.

NR/NT	5	10	25	50	75	100
1	1,03	1,04	1,03	1,03	1,02	1,05
5	2,61	2,38	1,90	1,41	1,22	1,07
10	3,44	2,60	1,53	1,09	0,93	0,96
25	3,38	2,63	1,51	0,94	0,86	0,97
50	2,54	1,57	1,14	0,83	0,83	0,89
75	1,85	1,18	0,85	0,84	0,83	0,89
100	1,54	1,11	0,80	0,86	0,84	0,88
150	1,32	1,03	0,87	0,84	0,86	0,91
200	1,28	1,04	0,90	0,87	0,88	0,92
250	1,19	1,02	0,92	0,87	0,87	0,92
350	1,06	1,00	0,94	0,86	0,89	0,93
500	1,13	0,98	0,93	0,86	0,87	0,94

Figure 72.  $E_{(c-jdbc,p,s)} / E_{(c-ctsa,p,s)}$  details.

**Update scenario**

The chart for the update scenario is shown in Figure 73. The comments made to the Select scenario are also applied to the Update scenario, regarding the transference rate of block of tuples. The most significant differences are: 1) the update protocol is a heavy protocol and, thus, its overhead has a deep impact on both entities and in the collected measurements; 2) the  $E_{(c-jdbc,p,\gamma)}$  entity has  $[nr]$  server cursors each one competing with the others to update the requested values while  $E_{(c-ctsa,p,\gamma)}$  entity has only one server cursor and the competition is performed at the client side. Despite the unfavorable conditions for C-CTSA, in this scenario, the ratio is always significantly greater than 1. It increases in the range  $1 < nt < 10$  and for  $nt > 10$  the ratios are practically stable for each individual  $[nr]$  (except for  $nr=5$ ). Another relevant issue is that the ratios decrease when  $[nr]$  increases for every  $[nt]$ .

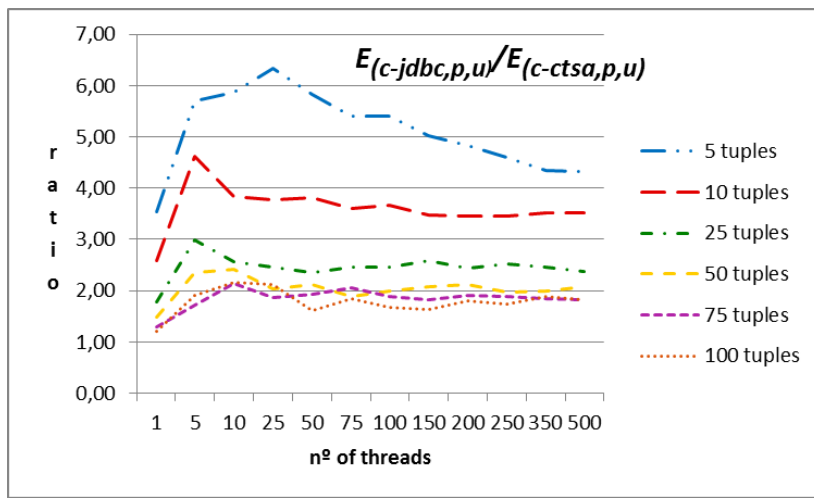


Figure 73.  $E_{(c-jdbc,p,u)} / E_{(c-ctsa,p,u)}$  chart.

**Insert scenario**

The chart for the insert scenario is shown in Figure 74. The most relevant aspect is the slight but

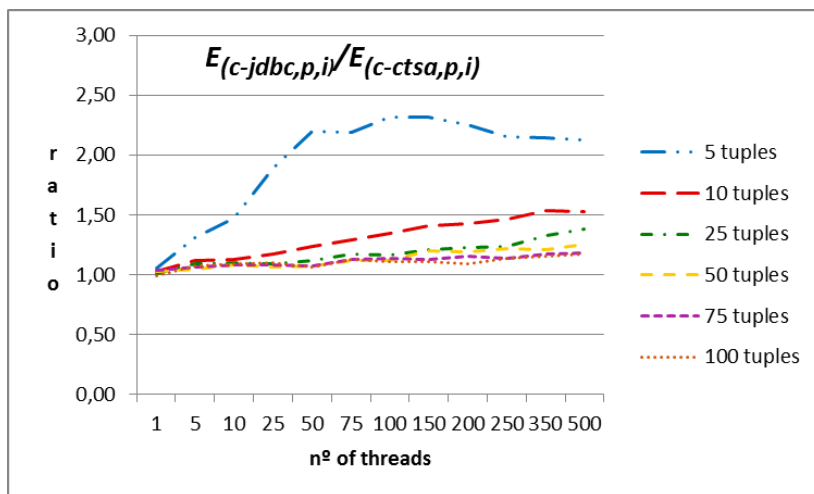


Figure 74.  $E_{(c-jdbc,p,i)} / E_{(c-ctsa,p,i)}$  chart.

constant ratios increase with  $[nt]$  for each  $[nr]$ , except for  $nr=5$ . In the initial stage, ResultSets empty and tuples are sequentially inserted and committed one by one in the host database table. In this scenario, in opposite to the others, all  $E_{(c-ctsa,p,\gamma)}$  threads insert adjacent tuples this way minimizing the number of blocks to be transferred. In spite of being a very heavy scenario for both entities, the differences between C-CSTA and C-JDBC are enough to be noticed in the ratios. It is always greater than 1 and higher values of  $[nr]$  cause a decreasing in the ratios.

## B.1.5 Conclusion

An architecture for a concurrent LMS of CLI, herein known as CTSA, has been presented. A proof of concept has also been presented based on a standard JDBC API. In order to assess CTSA performance in a concurrent environment and to compare it with an equivalent environment based on a standard JDBC solution, a test-bed has been defined and implemented with two concurrent entities: C-JDBC and C-CTSA. The measurements were collected using 3 scenarios. The scenarios were modeled to create favorable conditions to C-JDBC and unfavorable conditions to C-CTSA. This approach gives a much more secure perspective about the minimum expected gain in performance when using real scenarios. Thus, it is expected that when used in real scenarios, the gain in performance should be higher than the ones here presented and clearly bounded at the lower level by the ratios here presented. In spite of these adverse conditions, C-CTSA always gets better scores for the update and for the insert scenarios. In the Select scenario, C-CTSA obtained significant scores in the range of lower values of  $[nr]$  and  $[nt]$ . Anyway, for higher values of  $[nr]$  and  $[nt]$  the minimum ratio did not go below 0.8 which is still a remarkable score, considering the unfavorable conditions under which the assessment of C-CTSA took place.

The outcome of this research should encourage programmers of concurrent applications that use LMS of CLI, to implement a CTSA to improve the overall performance. Moreover, CLI providers should be encouraged to release CLI with embedded concurrency. Embedded concurrency should have the advantages of accessing the LMS's internal data structures to optimize the implementation of the different protocols. Very probably their results should be much better than the ones obtained through components derived on the CTSA as proved from the results obtained in [Gomes, '11].

## B.2 Embedded Approach

The embedded approach herein presented is based on a re-writing process of some parts of the original source-code of the ResultSet interface in order to make them thread-safe. The final document is available here [Gomes, '11] and, therefore, it will not be thoroughly described here. Only the key aspects are herein described and emphasized.

### B.2.1 Presentation

Whenever a scrollable or updatable LMS is instantiated, RDBMS create a database cursor. This one to one relationship between LMS and server cursors is not a scalable solution let alone in situations where many threads need to share the same data of the same Select expression. Thus, the basic idea is to transform the one to one relationship into a many to one relationship, that is, several LMS instances use the same server cursor. Another key issue is the internal operation of LMS. LMS have

an internal cache with a pre-defined size where the tuples are kept in memory. Whenever a client requests access to a tuple that is not in memory, LMS request the RDBMS to send the requested row. The RDBMS send a set of tuples, in accordance with the cache size, containing the requested tuple. This approach is efficient whenever the tuples are sequentially accessed but it is critical whenever the tuples are accessed randomly. The latter case means that, in an extreme scenario, whenever a tuple is requested, a new access to the server cursor is necessary.

## **B.2.2 Architecture**

Two solutions were implemented to address two common situations of concurrency between threads: each thread owns its private cache (individual cache) and all threads share the same cache of a LMS (shared cache).

### **B.2.2.1 Individual Cache**

The individual cache implementation provides several separated caches, each one containing a set of tuples accessible to one thread only. From the internal implementation point of view, this implementation does not promote concurrency but it is thread-safe.

#### Advantages

The individual cache presents two main advantages. Firstly, only one thread accesses the cache and, therefore, no concurrent mechanisms are necessary. The second advantage derives from the fact that when each thread needs no more tuples than those initially cached, the number of accesses to the server cursor is minimized.

#### Disadvantages

The individual cache presents three disadvantages. The first disadvantage has its origin on the need to copy tuples to each individual cache. The second disadvantage derives from the first and it is related to the existence of duplicated tuples. The third disadvantage also derives from the first and it is relevant when modifications occur in caches. The modified content is not immediately visible in other caches.

### **B.2.2.2 Shared Cache**

Unlike the individual cache, the shared cache implementation uses one cache only, which is shared by all threads. The shared cache has advantages and disadvantages next described:

#### Advantages

There are three main advantages which are the counterpart of the disadvantages of the individual cache implementation.

#### Disadvantages

Two main disadvantages are emphasized. The first one is the need to a thread-safe implementation of shared caches. The second disadvantage is the need for a new server cursor whenever a thread accesses a tuple not contained in the cache.

### B.2.3 Performance Assessment

The performance assessment was carried out to compare:

- the individual and shared caches;
- individual and shared caches against implementations based on a C-JDBC and also against the previous CTSA approach.

Several contexts were defined to see their impact on the overall performance. The tested contexts were:

- variable number of rows;
- variable number of threads;
- induced processing delays to simulate real scenarios - two scenarios were tested:
  - variable induced processing delays between consecutive attribute accesses following the next algorithm  
Access attribute 1  
Induced processing delay  
Access attribute 2  
....
  - Variable induce processing delays between consecutive row accesses following the next algorithm  
Next row {  
Access attribute 1  
Access attribute 2  
...  
Induced processing delay  
}
- Variable fetch size (number of fetched rows from server cursor to LMS). Were tested: 10%, 20%, 50%, 75% and 100%.

### B.3 Conclusion

The collected results have shown that the embedded approach leads to better performance results than the C-JDBC approach and the C-CTSA approach, see [Gomes, '11]. Thus, the embedded approach is a promising approach to implement a thread-safe LMS. The collected results would be even better if there was a deeper knowledge about the operation of server cursors at the time the work was done. As a final remark, providers of CLI should be encouraged to deploy thread-safe versions of their products.



## **Annex C – ABTC: Multi-purpose Adaptable Business Tier Components**

This research leverages previous work on Modelization, Componentization and Access Control of business tiers based on CLI Only the relevant aspects will be detailed to avoid the repetition of previously discussed and presented aspects.

### **C.1 Introduction**

CLI are general low level API that do not provide any high level assistance to address organizational and runtime needs. Three examples are provided:

#### Organizational needs

Some organizations decouple the development process of business tiers from the development process of application tiers. They are developed by different actors (people playing different roles). Unlike these organizations, others do not follow this separation of roles. The same person may be elected to play both roles.

#### Runtime needs

In some database applications, business tiers need to be dynamically adapted at runtime to address runtime needs. For example, to address evolving security needs or evolving business needs.

This gap is mainly derived from technical aspects of CLI previously addressed:

- source-code of business and application tiers is tangled and, therefore, the roles of programmers cannot be decoupled;
- CLI do not provide any means to adapt software to support different business needs, even if the CRUD expression is the same. Programmers have to re-write the same CRUD expression and re-write similar source code for the business tier part. This situation is critical in large database applications with many and complex CRUD expressions;
- CLI do not provide any access control mechanism.

To overcome these drawbacks of CLI, an architecture referred to as the Adaptable Business Tier Component (ABTC) is presented.

### **C.2 ABTC**

This section, in a first stage, presents and describes the architecture of ABTC. In a second stage, a proof of concept is also presented.

#### **C.2.1 Adaptation Process**

The adaptation process of business tiers is basically focused on the capability to support new CRUD expressions. It comprises two dimensions: the capacity to automatically build, when necessary, new

Business Entities and the capacity to accept and manage CRUD expressions. The first dimension is herein known as the service composition and the second is herein known as the service allocation. Beyond this, to address different adaptation needs, namely organizational and runtime needs, ABTC needs to support two lines: dynamic (ABTC\_Dynamic) and static (ABTC\_Static) versions. ABTC\_Dynamic is used when there is the need to carry out an adaptation process. ABTC\_Static is used when the adaptation process took place at an earlier stage and, therefore, there is no need to be carried out again. This approach conveys the need to persist the business logics involved in the adaptation process, whenever they are required in later stages. Persisted business logics are kept in independent components herein known as Business Logic. Thus, in this adaptation context (ABTC\_Dynamic, ABTC\_Static and Business Logic), three scenarios are presented to address organizational and runtime requirements, see Figure 75:

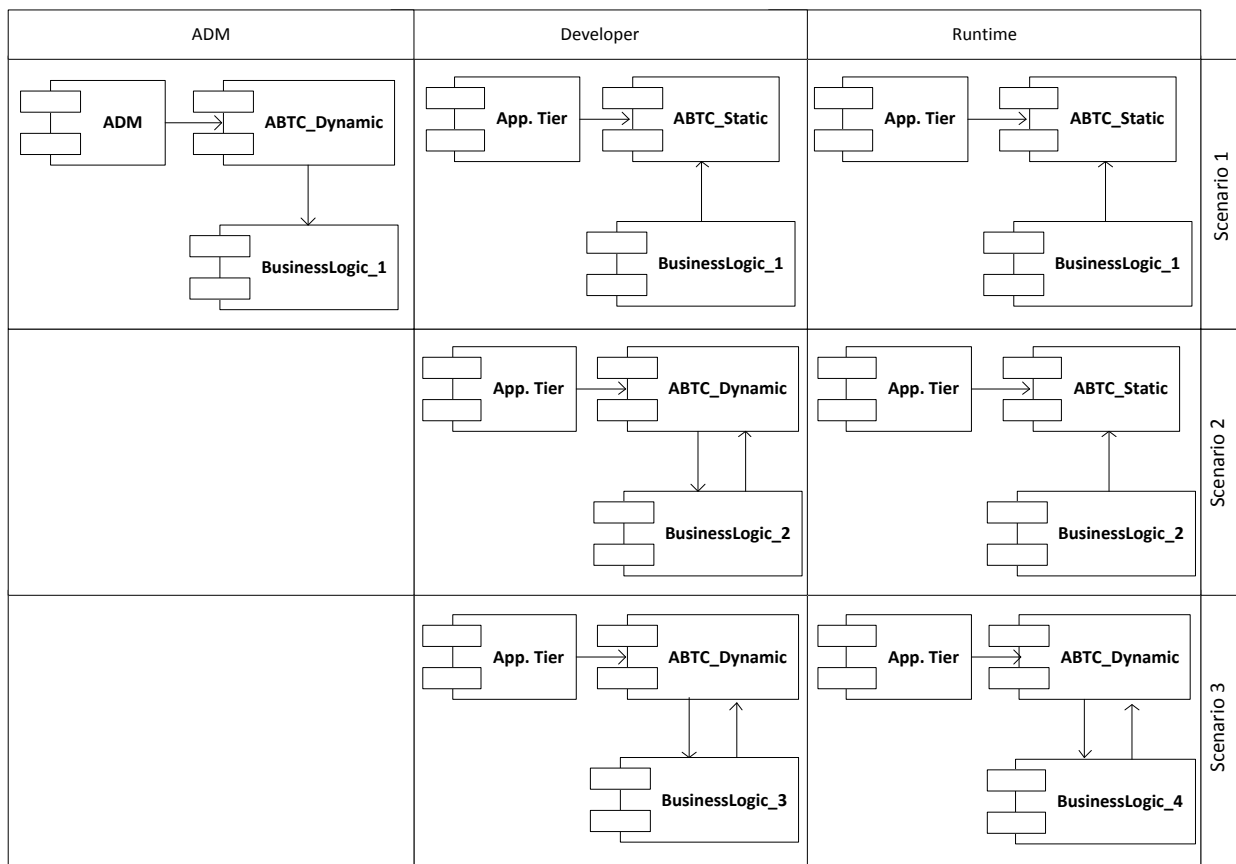


Figure 75. Implemented and tested scenarios.

**Scenario 1**

The adaption process is carried out by database administrators (ADM), or someone on their behalf. ABTC\_Dynamic is used to build a persistent business logic (BusinessLogic\_1). Then, developers of application tiers (Developers) use the persisted business logic (BusinessLogic\_1) and ABTC\_Static for the development process of application tiers and also for the database applications to be deployed (Runtime). This approach is used when business tier developer role (ADM) and application tier developer role (Developer) are played by different actors.

### Scenario 2

The adaptation process takes place during the development process of application tiers (Developer). It uses ABTC\_Dynamic to build a persistent business logic (BusinessLogic\_2). Then, database applications are deployed with the persisted business logic (BusinessLogic\_2) and ABTC\_Static. This approach is used when application tier developer role (Developer) and business tier developer role (Developer) are played by the same actor.

### Scenario 3

If necessary, business logics (BusinessLogic\_3) are built during the development process of application tiers (Developer) but are not used afterwards. Then, the adaptation process (BusinessLogic\_4) takes place after the deployment process of database applications and at runtime. This approach is used, for example, whenever the adaptation process is dynamic, eventually driven by security policies [Morin, '10] or eventually to accept CRUD expressions defined at runtime and already supported by existent CRUD schemas.

## **C.2.2 Architecture Presentation**

The architecture for ABTC is presented in this sub-section. The architecture is quite similar to the one presented for DACC. ABTC shares many concepts of DACA such as Business Schema, Business Entity, Service Composition and Service Allocation. Figure 76 presents the architecture for ABTC and only the differences for DACC are described.

### IServiceAllocation

IServiceAllocation comprises services to manage the service allocation process. The first two methods are used to manage the deployment process of CRUD expressions, CRUD by CRUD. The third method is used to deploy a set of CRUD expressions. While CRUD expressions deployed by the two first methods are not persisted within business logics, CRUD expressions deployed by the last method are persisted and replace all persisted CRUD expressions contained within the business logic.

### IServiceComposition

IServiceComposition comprises services to manage the service composition. The first two methods are used to manage the deployment process Business Schema by Business Schema. The third method is used to deploy a set of Business Schemas. In both cases, Business Services are persisted within Business Logics. The main difference is that Business Schemas deployed by the last method replace all previous Business Services within the business logic.

### IManager

IManager gathers services to provide one of the two supported versions: dynamic (1) or static (2) versions, ABTC\_Dynamic and ABTC\_Static, respectively. The dynamic version, beyond extending IServiceAllocation and IServiceComposition, comprises an additional method to define the repository for the persistent Business Logic.

### Manager

Manager provides two services:

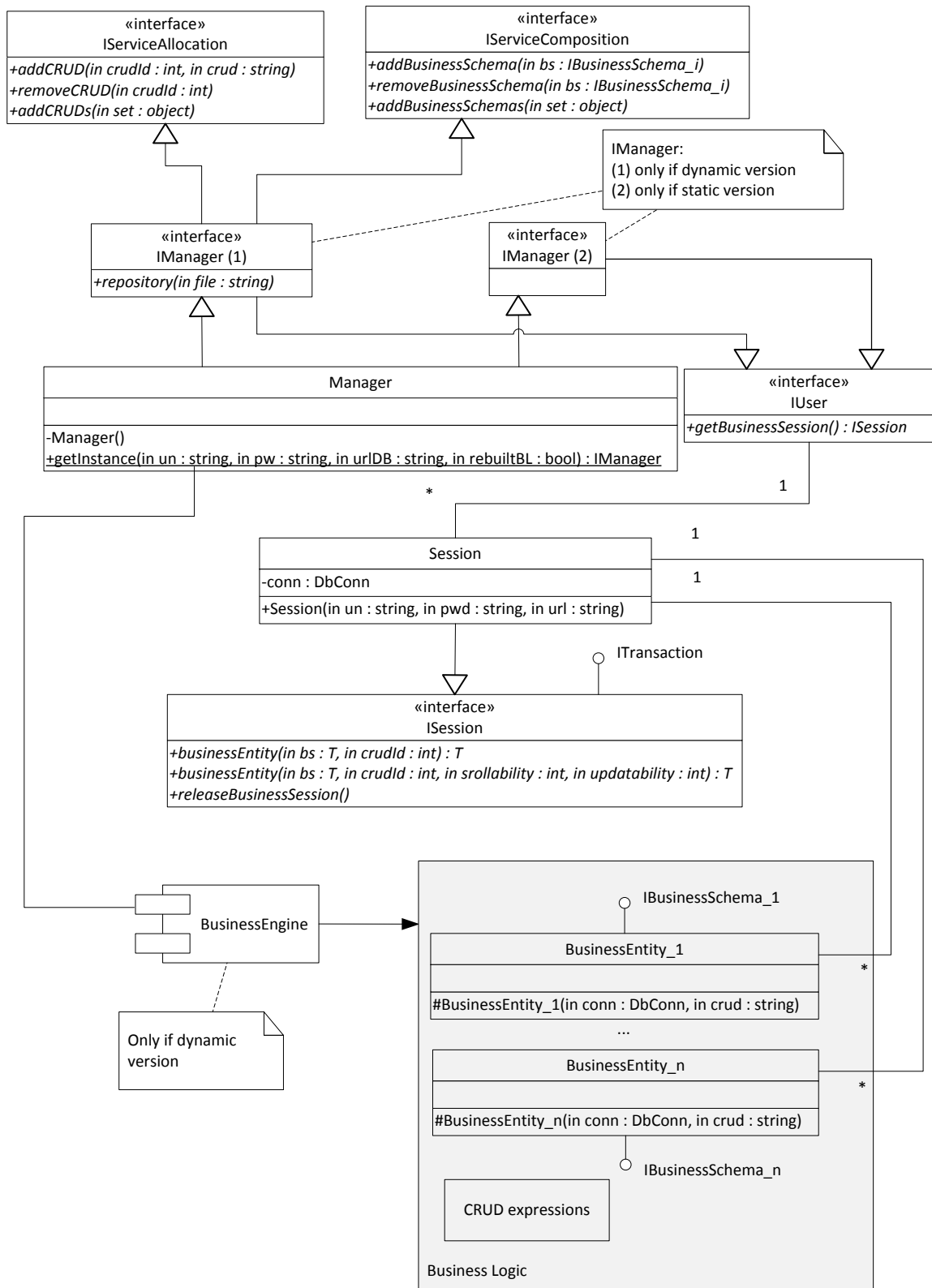


Figure 76. Class diagram of ABTC.

- a static method (*getInstance*) to create instances of ABTC;
- implements one of the two versions of IManager interface.

### Business Logic

Business Logic (at the bottom of the diagram) is an independent and persistent container to keep adapted business logics. Its content is updated through *IServiceAllocation* and *IServiceComposition* (only with the ABTC\_Dynamic).

### ISession and Session

ISession and Session, similarly to DACC, are responsible for managing the instantiation process of Business Entities. They provide three methods. The first two (*businessEntity*) instantiate Business Entities: the first one is only for Insert, Update and Delete expressions and the second one is only for Select expressions. They are generic methods that, among other arguments, accept a Business Schema and return an instance of a Business Entity that implements the requested Business Schema. To instantiate Business Entities, they need to be loaded into memory at runtime and, then, instances are created using reflection. This process ensures that Business Entities may be dynamically created and removed at runtime, without raising any runtime error. The second method opens the possibility to define functionalities of LMS at runtime (read-only or updatable and, forward-only or scrollable). This possibility cannot be in contradiction with the functionalities provided by the Business Entities being instantiated. For example, if a Business Entity is prepared to instantiate an updatable LMS, then the instantiated LMS may be used as read-only. The opposite, an LMS used as updatable but instantiated as read-only, raises a runtime exception. Sessions are released when not needed any more (*releaseBusinessSession*).

## **C.3 Proof of Concept**

This section presents the work carried out to prove that ABTC is a reliable architecture to overcome the presented drawbacks of CLI. As mentioned before, two components were built: ABTC\_Dynamic and ABTC\_Static. ABTC\_Dynamic implements the dynamic version and ABTC\_Static implements the static version.

Three demos are presented, one for each scenario, Figure 75, which will be herein used as the proof of concept. Demo1 is for scenario 1, Demo2 for scenario 2 and Demo3 for scenario 3. Demos are divided in two or three steps as shown in Figure 75: ADM, Developer and Runtime. Each demo comprises the same set of CRUD expressions and the same set of Business Schemas, presented in Table 12 (bottom line presents additional details to understand the table contents).

Tests were carried out with the three demos to evaluate if ABTC copes with the presented drawbacks of CLI. From the collected results, in the three demos, it is clear that ABTC completely addresses the goals defined for the three scenarios. ABTC\_Dynamic automatically builds Business Logic from Business Schemas and CRUD expressions. The Business Logic building process may be driven by any policy, being business needs and security policies only two different possibilities. ABTC\_Static uses Business Logic previously built with ABTC\_Dynamic.

These conclusions may be confirmed by accessing the public demos through the Windows Remote Desktop Connection (url: ned.av.it.pt; username: ABTC; password: guest).

ID	CRUD expressions	Business Schemas					LMS
		Closed				Open	
		IPrd_s	ICat_s	ISup_s	ICat_i	IOpen_s	
1	Select * from Products	Y	N	N	Y	Y	FR
2	Select * from Products where ProductID=10	Y	N	N	Y	Y	FR
3	Select * from Products where SupplierID=2	Y	N	N	Y	Y	FR
4	Select * from Categories	N	Y	N	Y	Y	FU
5	Select * from Categories where CategoryID=1	N	Y	N	Y	Y	FR
6	Select * from Suppliers	N	N	Y	Y	Y	FR
7	Select p.*, c.categoryName, c.Description from Products p, Categories c Where p.CategoryID=c.CategoryID	N	N	N	Y	Y	FR
8	Insert into Categories values (?, ?, ?, ?)	N	N	N	Y	N	
ID: CRUD expression identification (1-allFromProducts, 2-fromProducts_productId, 3-fromSuppliers_supplierId, 4-allFromCategories, 5-fromCategories_categoryId, 6-allFromCategories, 6-allFromSuppliers, 7-fromProductsCategories, 8-InsertInCategories CRUD expressions: supported CRUD expressions. Business Schemas: supported Business Schemas. LMS: updatability of LMS (F: forward-only, S:scrollable, R:read-only, U:updatable) User Perm.: permission for each user to use CRUD expression.							

Table 12. CRUD expressions and Business Schemas for the implemented scenarios.

A proof of concept is available through Windows Remote Desktop at: url: ned.av.it.pt, username: ABTC, password: guest. The three scenarios based on Figure 75 were implemented and Business Logic is defined from the contents of Table 12.

### C.4 Discussion

In this section a discussion is taken on the following aspects: 1) ABTC beyond JDBC; 2) IRead interface and 3) additional advantages of ABTC over CLI.

The proof of concept here presented is based on Java, JDBC and SQL Server 2008. An ABTC has also been built with C#, ADO.NET and SQL Server 2008. The component was manually built. The achieved success proved that the presented architecture is flexible enough to be used with different technologies. From our previous experience with O/RM tools, namely Java Persistence API, it is our belief that the architecture may also be used. However, it is so easy to be used with CLI that it would only bring disadvantages if used with O/RM, namely because of their induced overhead.

IRead interface is defined from schemas returned by Select expressions. Very often, these schemas derive directly from database schemas. This situation may raise several difficulties when a Select expression joins two or more tables having attributes with the same name, as it happens with Northwind. In this situation it is recommended to rename one of the attributes using ALIAS. Alias are useful, they increase the possibility of differentiating equal projected names from different tables. To avoid or minimize the usage of ALIAS, in order to minimize maintenance activities on IRead, we suggest the use of unique names for each attribute. If necessary, this may be achieved by using a technique based on a unique identification prefix for each table name and, then, using the

prefix to build attributes names in order to identify the source of each attribute. Example: Prd\_Products.Prd\_ProductId (table Products and one attribute ProductId).

In spite of not being key aspects of this research, we stress some additional CLI drawbacks which are also partially overcome by the reference architecture:

- With CLI, programmers need to master database schemas to deal with each retrieved attribute of each CRUD expression. With ABTC, IRead and IWrite interfaces provide schema-driven getter and setter methods, avoiding the need to master database schemas for each CRUD expression.
- With CLI, there is no easy way to link CRUD expressions and the applications they assist. With ABTC, the linkage is provided by schema-driven and type safe methods.
- ABTC, unlike CLI, transform runtime errors into compile errors. If the name of an attribute is modified (IRead is modified), new Business Services are built. Then, when the application tier is re-compiled, the compiler will detect all errors where the source-code of application tiers was not updated. With CLI, names of attributes are encoded inside strings, this way preventing any disconformity from being detected at compile time.

## C.5 Conclusion

CLI are used to build business tier components whenever performance is a key requirement. Regardless this advantage, they present some important drawbacks. To overcome the drawbacks, a multi-purpose architecture for ABTC is presented. Two versions of ABTC were defined to address different organizational and runtime needs. Three scenarios were defined and implemented as the proof of concept. It proved, among other issues, that ABTC address different organizational and contextual runtime needs. The adaptation process of ABTC is flexible to meet a wide set of different needs. The adaptation process relies on a two phase approach: the service composition, which takes place at runtime to dynamically build typed objects, and the service allocation, which also takes place at runtime to deploy CRUD expressions. This approach promotes the definition of different scenarios to address different needs. We have implemented three scenarios to address some organizational and contextual runtime needs. Other scenarios could be implemented, as for example, to address security needs. Basically, to address security needs, the deployment process of CRUD Schemas and CRUD expressions should be driven by access control policies. To promote the reuse of computation, Business Services manage not one but several CRUD expressions (closed and open approach).