Modeling And Inferring On Role-Based Access Control Policies Using Data Dependencies

Romuald Thion and Stéphane Coulondre

LIRIS: Lyon Research Center for Images and Intelligent Information Systems,
Bâtiment Blaise Pascal, 20 av. Albert Einstein, 69621 Villeurbanne Cedex, France
firstname.surname@liris.cnrs.fr

Abstract. Role-Based Access Control (RBAC) models are becoming a de facto standard, greatly simplifying management and administration tasks. Organizational constraints were introduced (e.g.: mutually exclusive roles, cardinality, prerequisite roles) to reflect peculiarities of organizations. Thus, the number of rules is increasing and policies are becoming more and more complex: understanding and analyzing large policies in which several security officers are involved can be a tough job. There is a serious need for administration tools allowing analysis and inference on access control policies. Such tools should help security officers to avoid defining conflicting constraints and inconsistent policies.
This paper shows that theoretical tools from relational databases are suitable for expressing and inferring on RBAC policies and their related constraints. We focused on using Constrained Tuple-Generating Dependencies (CTGDs), a class of dependencies which includes traditional other ones. We show that their great expressive power is suitable for all practical relevant aspects of RBAC. Moreover, proof procedures have been developed for CTGDs: they permit to reason on policies. For example, to check their consistency, to verify a new rule is not already implied or to check satisfaction of security properties. A prototype of RBAC policies management tool has been implemented, using CTGDs dedicated proof procedures as the underlying inference engine.

1 Introduction

DataBases Management Systems (DBMS) are cornerstones of Information Systems (ISs): they provide mechanisms to store, modify, retrieve and query information of an organization. In order to enhance security of data, Access Control (AC) mechanisms have been developed to manage users' rights over data stored in the DBMS. In its broader sense, AC, denotes the fact of determining whether a *subject* (process, computer ...) is able to perform an *operation* (read, write ...) on an *object* (a tuple, a table, ...). An operation right on an object is called *permission*. AC policies define the subjects' permissions.

Applications developed using DBMS can contain large amount of data with highly differentiated access for different users, depending upon their function or role within the organization [1]. Role-Based Access Control (RBAC) received

considerable attention as an alternative to traditional mandatory and discretionary AC policies in databases.

The RBAC models constitute a family in which permissions are associated with roles. A role is a job function or job title within the organization. Users are made members of appropriate roles. Permissions are not directly assigned to users (roles can be seen as collections of permissions) [2]. RBAC provides a powerful mechanism for reducing the complexity, cost, and potential for error in assigning permissions to users within the organization. RBAC was found to be among the most attractive solutions for providing AC in e-commerce, e-government or e-health [1, 3].

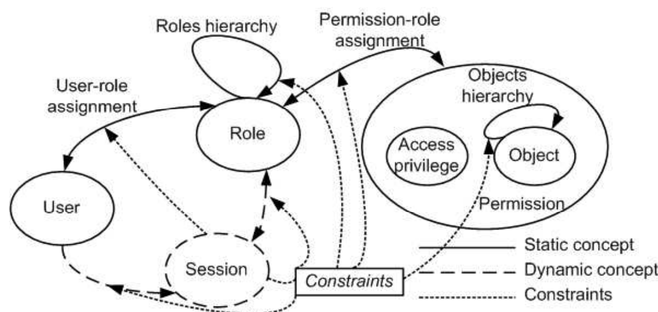

Fig. 1. RBAC Model

Nevertheless the number of users in RBAC policies is increasing and rules are more and more complex: diverse constraint [1] types have been introduced to reflect peculiarities of organizations. RBAC constraints specify conditions that cannot be violated by the components of the system.

Policies engineering is considered to be of high practical importance [4]: a large part of flaws in ISs are due to administration mistakes or security misconceptions. There is a need for tools facilitating design and maintenance of RBAC policies. According to the authors of [5], such tools need to be able to capture AC model mechanisms and peculiarities (e.g. RBAC constraints). These tools need to be able to check consistency of policies and to answer queries for particular permissions or relation holdings in the policies. Last requirement is a comprehensible inference mechanism, even by non-logicians. Our goal is to provide a formal framework satisfying these requirements.

[1] *constraint* may be a confusing word in this paper: it may either designate relations between variables (e.g., $X \leq Y, X \geq 2 \times Z + 1, 3 = T, 2 \neq 3$, etc.), restrictions on RBAC model's concepts (e.g. nobody is allowed to assume simultaneously roles r_1 and r_2) and even data dependencies (integrity constraints). In this paper we do not use the term *integrity constraints*, *constraints* refers to semantic relations between variables and *RBAC constraints* or *organizational constraints* refers to restriction among elements in RBAC policies.

Thus our contribution is twofold:

- identification of a theoretical tool from the databases field suitable for homogenous modeling of RBAC principles and its related constraints right into the relational model,
- use tools built (e.g. proof procedures) on top of the underlying theoretical model to provide a set of tools facilitating design and management of RBAC policies in order to detect and correct administration mistakes or misconceptions.

The class of dependencies we focused on is Constrained Tuple-Generating Dependencies (CTGDs). CTGDs is among the widest class of dependencies [6]. We will show that a this framework is an appropriate formal tool for representing and checking RBAC policies.

In the next section we will introduce CTGDs and proof procedure related in. Section 3 will show how CTGDs can be used to model RBAC concepts, constraints and assignments, of which implements will be shown in section 4. Section 5 will summarize some papers related to this work. Finally, last section will discuss our work and presents perspectives using databases dependencies for security purposes.

2 Background

2.1 Constrained Tuple-Generating Dependencies

The authors of [6] expose a kind of data dependencies upon the most expressive existing: CTGDs, generalizing traditional dependencies such as FDs, MVDs or EGDs. CTGDs extend Tuple-Generating Dependencies (TGDs, which are also known as *Generalized Dependencies* [7,8]) with a constraint domain (e.g. linear arithmetic over integers, rationals or real). Constraints are quite interesting when used jointly with existential quantifiers because they permit a more precise definition of such *partially known* facts. CTGDs are interesting for DBMS storing complex data such as spatial, audio, image, video or temporal data. CTGDs can be represented formally in First-Order Logic (FOL) by formulae of the form [7]:

$$\forall X\ p_1(X_1) \wedge \ldots \wedge p_i(X_i) \wedge c(X) \rightarrow \exists Y\ q_1(X \cup Y_1) \wedge \ldots \wedge p_j(X \cup Y_j) \wedge c'(X \cup Y)$$

where p_i and q_j are predicates symbols, $X = X_1 \cup \ldots \cup X_i$ is the set of all terms (no functions symbols) in the left hand side. Terms of X are universally quantified. $Y = Y_1 \cup \ldots \cup Y_j$ does not designate all terms in the right hand side, but only those that are not bound by the universal quantifier on the left hand side. Terms of Y are existentially quantified. Finally, c and c' are conjunctions of linear constraints ($<, >, \leq, \geq, \neq, =$) over terms (terms of X for c and terms of $X \cup Y$ for c').

For example, the following dependency from [6] expresses that cpath(source, destination, cost), which contains the cheapest path between any two points in a directed graph with edge weights, is a transitive closed relation obeying the triangle inequality: $\forall S_1, D_1, C_1, D_2, C_2\ cpath(S_1, D_1, C_1) \wedge cpath(D_1, D_2, C_2) \rightarrow \exists C_3\ cpath(S_1, D_2, C_3) \wedge C_3 \leq C_1 + C_2$.

2.2 Dedicated Proof Procedures for CTGDs

The authors of [6] propose two bottom-up chase over CTGDs. Their paper addresses the implication problem, that is, given a collection of CTGDs F, and a single CTGD g, determine whether in every database state where F is satisfied, it is also the case that g is satisfied. The chase proves if F logically implies g, stated briefly as $F \vDash g$.

The operational nature of these proof procedures is based on the concept of tuple (a grounded atom, with no variables). Basic outline of such procedures is based on [9] with the adjunction of constraints: hypothesize the existence of some tuples in the relations such that the antecedent l of g is satisfied, treat F as defining a closure operator generating tuples $F(l)$. On each computation step of $F(l)$, the following condition is tested:
- if $F(l)$ contains a copy of r, infer $F \vDash g$,
- if $F(l)$ contains an inconsistency produced by constraints, infer $F \vDash g$ vacuously,
- if $F(l)$ does not contain a copy of r, infer $F \nvDash g$.

The CTGDs implication problem is semi-decidable: the procedures may run forever. As each basic step is producing new facts through implication, we can practically bound up the number of successively applied CTGDs (e.g. to avoid circular generating facts leading to infinite loop), but it is unsound and must be reserved for implementation purpose. However, there exist decidability results for specific subclasses of CTGDs. For example, the chase is decidable for Full-TGDs, TGDs without existentially quantified variable [9].

3 A Framework For Expressing And Checking RBAC Policies

According to the authors of [10] we use the following predicates to model core concepts of RBAC policies:
- $ura(User, Role)$, to define User Role Assignments,
- $pra(Access, Object, Role)$, to define Permission Role Assignment
- $permitted(User, Access, Object)$, to specify that user $User$ is granted $Access$ access privilege on object $Object$.

3.1 Capturing axiomatic definition of RBAC model

Once basic elements of the policies are defined, we need to model the "axiomatic of RBAC": the core of the AC model which settles how an access is granted to a user through role assignment and how is defined hierarchy. We model an RBAC axiomatic based on [10]. $dSenior(SeniorRole, JuniorRole)$ to define direct inheritance between roles and $senior(SeniorRole, JuniorRole)$ to define role hierarchy (the transitive closure of $seniorDirect$).
- role inheritance is transitive: $senior(X, Y), dSenior(Y, Z) \rightarrow senior(X, Z)$,
- role inheritance is irreflexive: $senior(X, X) \rightarrow \bot$,
- a user is access granted to an object if he is assigned to a role which is assigned

to this permission: $ura(U, R), pra(A, O, R) \rightarrow permitted(U, A, O)$,
- eventually through inheritance $ura(U, R1), senior(R1, R2), pra(A, O, R2) \rightarrow permitted(U, A, O)$.

3.2 Capturing Organizational Constraints

Constraints are an important aspect of RBAC and are a powerful mechanism for laying out higher-level organizational policy [2]. The best known RBAC constraints are:

Mutually exclusive roles constraints settle that no user can be assigned two roles which are in conflict with each other. In other words, it means that conflicting roles cannot have common users. $ssd(Role1, Role2)$, specificy that $Role1$ and $Role2$ are in Static Separation of Duties (SSD): they are mutually exclusive. Mutually exclusive roles can produce inconsistency. The authors of [11] describe a set of properties that must hold in any RBAC policy. These properties are described in the example of section 4.

Cardinality constraints settle that a number of assignments is limited. Cardinality constraints of n maximum users assigned to role r can be expressed in CTGDs by $\wedge_{i=1}^{n+1} ura(U, N_i)\ \{\forall i \in [1..n], \forall j \in [i+1..n+1]\, N_i \neq N_j\} \rightarrow \bot$. Mutually exclusion and cardinality constraints are not limited to role and can be used on any element of the policy model (for example with access: no role can be granted both read access and write access on an object o). Our approach can be generalized for maximum number of roles assigned to users or to permissions.

More generally, Nullity Generating Dependencies of the form $p_i(X) \wedge c \rightarrow \bot$ can be used to model RBAC constraints: an RBAC constraint define that if a certain state (the left hand side of the CTGD) is reached, then the policy is inconsistent (right hand side is $\bot$).

Prerequisite constraints settle that if a particular relation holds, another holds too. Variables appearing only within the terms of the tail in CTGDs are existentially quantified. Intuitively that does mean *at least one element such as ... exists*. This semantic is used to take into account prerequisite RBAC constraints. E.g. role r_2 is required by role r_1: *for any user assigned to role* r_1*, at least one another user must be assigned to role* r_2, $ura(U_1, r_1) \rightarrow ura(U_2, r_2)\ U_1 \neq U_2$. Other prerequisite constraints can be expressed using CTGDs, according to administrator's need. CTGDs can model other forms of prerequisite constraints on any RBAC concept.

3.3 Inference on Policies

Depending on which stage of the RBAC specification one is working on, different needs of verification may exist:
- during the stage of modeling axiomatic (the core policy model), we are likely to check the expected behavior of the model and rules redundancies. E.g. how authorizations are derived from user-role and permission-role assignment,
- during the stage of defining the role hierarchy, we are likely to check a set of properties. E.g. there is no cycle in the hierarchy, or no role inherits the

administrator role,
- during the stage of defining user-role and permission-role assignment we are likely query the policy and to check a set of properties. E.g there is no two roles which have exactly the same permissions,
- during the stage of defining constraints it is interesting to check whether the policy is consistent, in other words if we settled facts violating constraints.

Security requirements	Reduction into CTGDs
Security property that must hold in all RBAC policy instances. *no role can be senior to itself*	Model the property to verify by a single CTGD and use proof procedure to check implication from axiomatic of RBAC and organizational constraints
Check if a policy is consistent	Try to derive $\perp$ from the policy by proof procedure
Security property that must hold in a policy instance. *no role inherits the administrator role*	Model the property to verify by a single CTGD and verify if it is satisfied by the database instance
Policy management capabilities: queries and data manipulation *which users are assigned to role student?*	Process query over the database

Table 1. Reduction of security administration needs into CTGDs-dedicated tools

The second termination case (vacuously) of algorithms from [6] is very useful while checking AC policies, it denotes that the policies are inconsistent. This semantic is interesting for security administrators when dealing with constrained AC policies: if there are facts violating constraints, the policy is inconsistent.

4 Experimental Validation

This section illustrates how a RBAC policy can be modeled into CTGDs. The sample code is separated into four parts: the first one models the core mechanisms of the RBAC model and settles a set of properties that must holds in any RBAC policy [11]. The second part is a sample role hierarchy used in a virtual organization. Unfortunately we are limited to toy sample or randomly generated policies, because organizations are not likely to share such sensitive information. Next is a sample definition of User-Role Assignments and Permission-Role Assignments. The last part defines a set of specific organizational constraints that must hold in this particular policy.

```
%axiomatic definition of RBAC policies and generic constraints
%------------------------------------------------------------
%senior is the transitive closure of dSenior
dSenior(SeniorRole,JuniorRole)->senior(SeniorRole,JuniorRole).
senior(SeniorRole,InterRole), dSenior(InterRole,JuniorRole)-> senior(SeniorRole,JuniorRole).
senior(Role,Role)->false.
```

```
%granting access to user through role assignments
ura(User,Role),pra(Access,Object,Role)->permitted(User,Access,Object).
ura(User,SeniorRole),senior(SeniorRole,JuniorRole),
pra(Access,Object,JuniorRole)->permitted(User,Access,Object).

%Property P1: any two roles assigned for a same user are not in separation of duties
ura(User,Role1),ura(User,Role2),ssd(Role1,Role2)->false.

%Property P2: no role is mutually exclusive with itself
ssd(Role,Role)-> false.

%Property P3: mutual exclusion is symetric
ssd(Role1,Role2)->ssd(Role2,Role1).

%Property P4: any two roles in ssd do not inherits one another
senior(Role1,Role2),ssd(Role1,Role2)->false.

%Property P5: there is no role inheriting to roles in ssd
ssd(Role1,Role2),senior(SeniorRole,Role1),senior(SeniorRole,Role2)->false.

%Property P6: If a role inherits another role and
%that role is in SSD with a third one, then the inheriting
%role is in SSD with the third one.
ssd(Role1,Role2),senior(SeniorRole,Role1)->ssd(SeniorRole,Role2).

%definition of role hierarchy
%----------------------------
%roles and hierarchy (with directly senior predicate) modeling
->role(student),role(researcher),role(teacher),role(phDStudent).
->role(postPhD),role(lecturer),role(seniorLecturer),role(professor).
->dSenior(phDStudent,student), dSenior(phDStudent,researcher).
->dSenior(postPhD,phDStudent), dSenior(postPhD,teacher).
->dSenior(lecturer,teacher), dSenior(lecturer,researcher).
->dSenior(professor,seniorLecturer), dSenior(seniorLecturer,lecturer).

%definition of assignments
%------------------------
%Permission-Role Assignments
->pra(read,test,student),pra(write,test,teacher),pra(read,finalTest,professor).
->pra(read,smallPaper,lecturer),pra(write,bigPaper,professor).

%User-Role Assignments
->ura(alice,student),ura(bob,phDStudent),ura(charly,professor).

%definition of organizational constraints
%---------------------------------------
%prerequisite on permissions: if one can read and object, another one can write
pra(read,Object,Role1) -> pra(write,Object,Role2) {Role1=\=Role2}.

%uniqueness constraint on manager
ura(User1,manager),ura(User2, manager){User1=\=User2}->false.

%mutually exclusives roles: student and professor
->ssd(student,lecturer).
```

We have described chase procedures as algorithms proving that a set of CTGDs F implies a single CTGD g: $F \vDash g$. The above ruleset is such an F collection, and g is the security property to check. The table 1 describes how tools dedicated to CTGDs can be used by administrators to design, verify and manage their policies. Six properties ($P1$ to $P6$) are settled in the sample policy, the authors of [11] have manually demonstrated the following theorem: $P2 \wedge P3 \wedge P6 \Rightarrow P4 \wedge P5$.

Our first example illustrates how chase procedures for CTGDs can be used to automatically proove the same theorem:
- let F be the collection of CTGDs modeling properties $P2$, $P3$ and $P6$,

- let be g_1 the CTGD modeling properties $P4$,
- let be g_2 the CTGD modeling properties $P5$.

The chase procedure prove that $F \models g_1$ and $F \models g_2$, we can conclude the properties $P4$ and $P6$ are redundant. Such functionalities are very interesting for security administrators: they can check that security properties ($P4$ and $P6$ in this example) hold in all RBAC policy instances (that satisfy $P2$, $P3$ and $P6$ in the example).

Another example is $g \equiv ura(joe, student), ura(joe, seniorLecturer) \rightarrow$: "is the policy consistent if *joe* is assigned to both *student* and *seniorLecturer*?". Clearly, with such assignments to user *joe*, the policy is inconsistent: roles *student* and *lecturer* are in SSD, *student* and *seniorLecturer* are in SSD too according to property $P6$. thus the policy is inconsistent using property $P1$. It is very interesting for administrators to conduct such verifications *before* any assignment: they can ensure the consistency of their policy in the presence of updates.

We have implemented a toolkit, "TGDToolBox", written in C++ to provide a set of functions to deal with data dependencies (e.g. syntatic analysis, unification of atoms, variables renaming). We implemented the chase procedures described in [6] using this library to run examples from this section. Actually, the prototype is able to handle hundreds of CTGDs and to answer in interactive time. We are currently using the toolkit to develop new proof procedures for dependencies [8, 12]. Using the proof procedures as an inference engine, we have built a proof of concept Microsoft Visio 2003 Template dedicated to RBAC policies design. This template provides an iconic interface for RBAC policy management. It is able to determine if a permission is granted to a user through his role assignment, it can check if the set of policies is consistent and can answer queries about the relations holding in the RBAC policy.

5 Related works

Our work has been influenced by [10] which express RBAC models with Constraint Logic Programming and [13] which describes the "Flexible Authorization Framework", that can be analyzed using a variant of Datalog (typically either safe stratified Datalog or Datalog with constraints).

The three main arguments we focused on are providing a framework which:
- is able to capture all relevant concepts of RBAC models,
- can benefit researches (e.g. evolutions, theoritical results, implementations) from a well established community,
- can be easily linked with other components of ISs (e.g. databases).

The authors of [10] describe AC programs able to deal with RBAC models. This very complete work addresses many problems arising with the use of closed policies (access denied as a default action, authorizations are only ever positive), open policies (access granted as a default action) or hybrid policies (authorizations and denial can be explicitly defined). However, logical programs

are not intuitive for non-specialits and the logic used do not integrate existential quantifiers. Moreover, RBAC policy are already widespread, a framework base on databases makes integration of administration tools and security data easier.

The autors of [4] argue"'... extensive research activity has resulted in the definition of a variety of AC ... Thus, the need arises for developing tools for reasoning about the characteristics of these models. These tools should support users in the tasks of model specification, analysis of model properties, and authorization management". Their logical framework is based on the C-Datalog language, whereas our is based on CTGDs, which is a able to deal with a wider class of rules thanks to existential quantifiers and constraints within both head and tail of dependencies.

The authors of [14] describe a fragment of FOL which tractable and sufficiently expressive to capture policies for many applications. This work is really interesting and points out tractability and complexity results on their logic. Constraints in policies are necessary to capture peculiarities of organizations, but modeling such restrictions is not develop in [14]. We do agree the authors statement about the use of logic programming by non-logicians, but we disagree that a "filling the blank on English sentences interface is sufficient for security administrators. We think that administrators must have a computer-aided software engineering (CASE) interface to design and check policies and such a CASE should provide a comprehensible trace of reasoning.

6 Conclusions and further work

We are confident that CTGDs can be used to express other AC models such as Task-BAC, Workflow-BAC, Mandatory-AC or Organization-BAC. Our fragment of FOL is really closed to the ones used in [10] or [4], which are able to deal with temporal aspects and at least mandatory and discretionary AC models.

For sake of clarity the example exposed in section 4 does not include sessions. According to [10] sessions and dynamic constraints can be captured easily with CLP. We are investigating the interest of chase procedure to check RBAC policies involving sessions. For example, using chase procedure we might answer queries like *Are the policies consistent for all possible sessions?*. Moreover, incorporating the model for administration of roles exposed in [15] is promising for distributed policies verification purposes.

Integrating of temporal aspects in RBAC models has been investigated in [16]. The authors of [10] use the Constraint Logic Programming framework. We can use the same approach to model Temporal-RBAC models, and according to [17] we extend the inequalities to geographical trigerring of assignments. Integrating temporal or geographical concerns into CTGDs, is mainly related to the choice of a right *constraint domain* [6]. For example, to define that a role is assigned to a user only on [t1, t2] shift, (between the times t1 and t2): $time(H)\{t1 \leq H \leq t2\} \rightarrow ura(user, role)$.

A promising opening to the use of CTGDs for AC modeling purpose is the result exposed by the authors of [12]. They propose a new kind of dependencies

subsumming CTGDs : Disjunctive-CTGDs. Their enhanced expressivity can be used to model new kinds of organizational constraints involving disjunctions, classes of constraints which have not been studied in the AC literature yet.

References

1. Ramaswamy, C., Sandhu, R.: Role-based access control features in commercial database management systems. In: Proc. 21st NIST-NCSC National Information Systems Security Conference. (1998) 503–511
2. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Computer **29**(2) (1996) 38–47
3. CERT/CC, U.S.S., magazine, C.: E-crimewatch survey. Technical report, http://www.cert.org/archive/pdf/ecrimesummary05.pdf (2005)
4. Bertino, E., Catania, B., Ferrari, E., Perlasca, P.: A logical framework for reasoning about access control models. ACM Trans. Inf. Syst. Secur. **6**(1) (2003) 71–127
5. Bonatti, P.A., Samarati, P.: Logics for authorization and security. In Chomicki, J., van der Meyden, R., Saake, G., eds.: Logics for Emerging Applications of Databases, Springer (2003) 277–323
6. Maher, M.J., Srivastava, D.: Chasing constrained tuple-generating dependencies. In: PODS, ACM Press (1996) 128–138
7. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
8. Coulondre, S.: A top-down proof procedure for generalized data dependencies. Acta Inf. **39**(1) (2003) 1–29
9. Beeri, C., Vardi, M.Y.: A proof procedure for data dependencies. J. ACM **31**(4) (1984) 718–741
10. Barker, S., Stuckey, P.J.: Flexible access control policy specification with constraint logic programming. ACM Trans. Inf. Syst. Secur. **6**(4) (2003) 501–546
11. Gavrila, S.I., Barkley, J.F.: Formal specification for role based access control user/role and role/role relationship management. In: ACM Workshop on Role-Based Access Control. (1998) 81–90
12. Wang, J., Topor, R., Maher, M.: Reasoning with disjunctive constrained tuple-generating dependencies. Lecture Notes in Computer Science **2113** (2001) 963–973
13. Jajodia, S., Samarati, P., Sapino, M.L., Subrahmanian, V.S.: Flexible support for multiple access control policies. ACM Trans. Database Syst. **26**(2) (2001) 214–260
14. Halpern, J.Y., Weissman, V.: Using first-order logic to reason about policies. In: CSFW, IEEE Computer Society (2003) 187–201
15. Sandhu, R.S., Munawer, Q.: The arbac99 model for administration of roles. In: ACSAC, IEEE Computer Society (1999) 229–240
16. Bertino, E., Bonatti, P.A., Ferrari, E.: Trbac: A temporal role-based access control model. ACM Trans. Inf. Syst. Secur. **4**(3) (2001) 191–233
17. Grumbach, S., Rigaux, P., Segoufin, L.: Spatio-temporal data handling with constraints. GeoInformatica **5**(1) (2001) 95–115