# Obtaining Secure Code in SQL Server Analysis Services by using MDA and QVT

Carlos Blanco[1], Ignacio García-Rodríguez de Guzmán[1], Eduardo Fernández-Medina[1], Juan Trujillo[2] and Mario Piattini[1]

[1] Dep. of Information Technologies and Systems. Escuela Superior de Informática
Alarcos Research Group – Institute of Information Technologies and Systems
University of Castilla-La Mancha, Paseo de la Universidad, 4. 13071. Ciudad Real, Spain

[2] Department of Information Languages and Systems. Facultad de Informática
University of Alicante. San Vicente s/n. 03690. Alicante, Spain

**Abstract.** Data Warehouses manage historical information for the decision making process that could be found out by unauthorized users when security constraints are not established. Therefore, it is very important for OLAP tools to consider the security rules defined at early stages of the development lifecycle. Following the MDA approach we have created an architecture for developing secure Data Warehouses and in this paper we complete this architecture obtaining secure multidimensional code in SQL Server Analysis Services from our secure multidimensional conceptual model (SECDW) by using QVT transformations. We focus on automatically obtain code for the security constraints defined at upper abstraction levels.

## 1 Introduction

Multidimensional modeling is the foundation of Data Warehouses (DWs), multidimensional (MD) Databases and On-Line Analytical Processing Applications (OLAP). Data Warehouses systems are used by decision makers to analyze the status and the development of an organization [1], based on large amounts of data integrated from heterogeneous sources into a multidimensional (MD) model.

On the other hand, information security is a serious requirement that must be carefully taken into account, not as an isolated aspect, but as an element present in all stages of the development lifecycle: from requirement analysis to implementation and maintenance [2, 3]. In this way, information assurance, security and privacy have moved from being considered by information systems designers as narrow topics of interest to become critical issues of fundamental importance in our society [4]. Some authors indicate that the survival of organizations depends on the correct management of information security and confidentiality [5]. Data Warehouses use enterprise information for the decision making process and a user can find out very important information by using queries in OLAP tools. In this way, it is necessary that security measures defined in all early stages of the development process are applied in OLAP.

DOI: 10.5220/0001745200380048

In addition, OMG Model Driven Architecture (MDA) [6] is an standard for model-driven approaches for software development that is based on the separation between the specification of the system functionality and its implementation using specific platforms. MDA defines a *Platform-Independent Model* (PIM) that does not include information about specific platforms and technologies. This model (PIM) can be translated into: (1)one or more platform-specific models (PSM) with information about the used specific technology; or (2) other PIMs with a different level of abstraction. Then, each PSM can be translated into a code that can be executed in the specific platform. There are several proposals for defining these translations between models [7], and OMG proposes to use Query/Views/Transformations (QVT) [8] for defining transformations between models created by using Meta-Object Facility (MOF).

In [9] a proposal for modeling secure Data Warehouses using a MDA approach, that will be described in the following section, is presented. This proposal does not deal with the final implementation into OLAP tools. [10] discuss how security measures defined by using this approach could be finally implemented into OLAP tools. In this paper, authors focus on obtain secure multidimensional code for SQL Server Analysis Services in an automatic way following the MDA approach. Due to the lack of space (and the extent of the presented MDA transformation), the QVT rules dealing with the generation of the structural aspects are not presented, but those implementing the security issues.

The rest of the paper is organized as follows: Section 2 we will describe our MDA approach for developing secure DWs; Section 3 we will present our QVT transformations; and finally, Section 4 we will present our conclusions and future work. Code for the proposed rules will be presented in greater detail in Appendix.

## 2 Model Driven Architecture For Developing Secure DWS

In this section our Model Driven Architecture for developing secure Data Warehouses is presented. We focus on the description of the source (Secure Multidimensional PIM) and target (SSAS Metamodel) models used by the QVT transformations proposed in this work. Figure 1 illustrates our MDA architecture for developing secure Data Warehouses [9]. In this architecture security constraints specified at upper abstraction levels are translated into conceptual, logical and code levels.

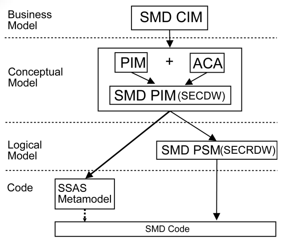


**Fig. 1.** MDA approach for developing secure DWs.

At business level we define both functional and non functional requirements for DWs by using an UML profile for i* called Secure Multidimensional CIM (SMD CIM) [11]. Using a QVT transformation it is possible to obtain a Secure Multidimensional PIM (SMD PIM) at conceptual level represented with our Secure Data Warehouse (SECDW) metamodel [12] that is an UML profile for DWs enriched with an Access Control and Audit model [13]. This conceptual model will be the source model for the proposed QVT transformations and will be explained in more detail.

The specification of a platform-specific model (PSM) is designed according to the specific properties of the Database Management Systems (DBMS) such as Relational Online Analytical Processing (ROLAP), Multidimensional Online Analytical Processing (MOLAP) or Hybrid Online Analytical Processing (HOLAP). We have defined a Secure Multidimensional PSM (SMD PSM) at logical level, that is a ROLAP approach called Secure Relational Data Warehouse (SECRDW) [14]. This metamodel extends the Relational Warehouse Metamodel (CWM) with security and audit capabilities and allows us to model STables, SColumn, PrimaryKey, ForeingKey, etc. The SecurityProperty and SecurityConstraint metaclasses are associated with the Table and Column metaclasses to allow us defining security at table and attribute levels. Besides, SecurityConstraints let us express the constraints (SecurityRule, AuthorizationRule and AuditRule) that are defined in the SECDW metamodel with UML notes.

At code level we obtain metamodels with secure multidimensional code in the target platform that can be easily translated into final code. In this work we define several metamodels to represent constraints over a multidimensional approach (MOLAP) in SQL Server Analysis Services. These metamodels will be the target models for the proposed QVT transformation and will be presented in more detail in following sections.

In order to complete the MDA architecture, it is necessary to define the transformation between models. The transformation between conceptual and logical levels using SMD PIM (SECDW) and SMD PSM (SECRDW) has already been defined [15]. Furthermore, the transformation from our relational metamodel at logical level (SECRDW) to secure code in a DBMS using Oracle Label Security has been defined too.

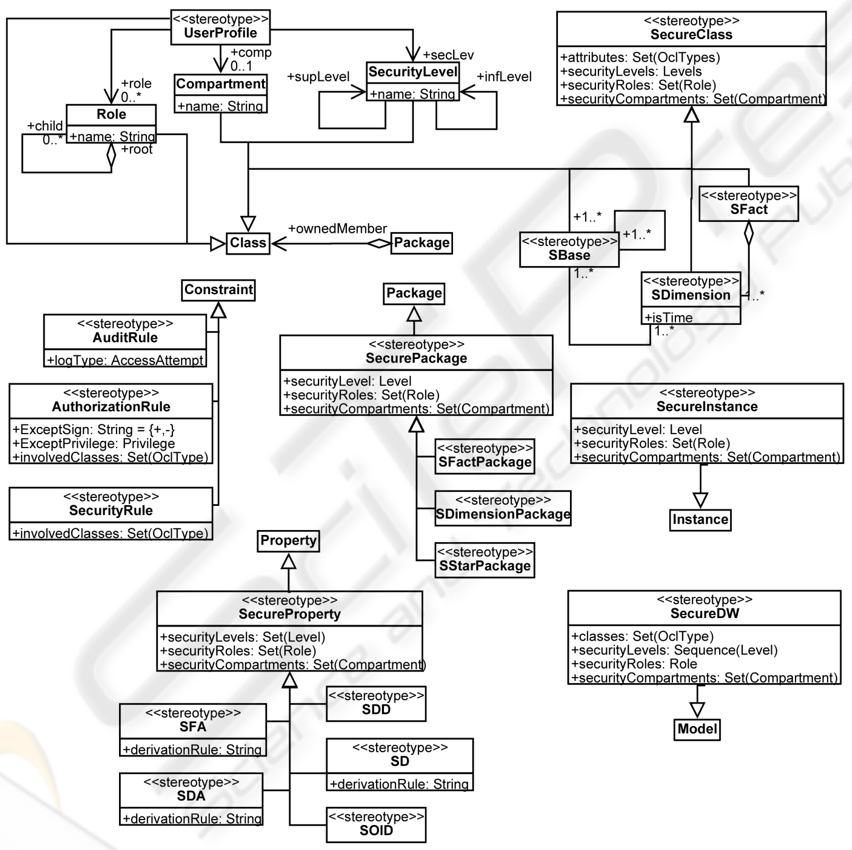
Due to the fact that in Data Warehouses OLAP tools are more used than DBMS, our research effort is focused on developing multidimensional secure code into OLAP tools according to the above-defined security requirements at conceptual and logical levels. In this paper, we follow the methodology defined in [10] to translate security constraints defined at conceptual level (SECDW) into secure multidimensional code in SQL Server Analysis Services, that can be automatically translated into final code.

### 2.1 Secure Multidimensional Pim (SECDW)

Our Secure Multidimensional PIM, called Secure Data Warehouse (SECDW) [12], allows us to represent the main security requirements for DW at conceptual level and it is composed of an UML profile for DWs [16] enriched with an Access Control and Audit (ACA) model [13].

Traditional access control models are based on relational concepts (tables, columns, rows, etc) and they are not appropriate for the multidimensional modeling used in Data Warehouses. The ACA model [13] is an access control and audit model defined for DWs that allows us to specify security constraints in DW's multidimensional models.

This model considers a combination of mandatory and role based access control which is based on the classification of subjects and objects in the system. ACA defines three ways of classification: security levels that indicate the clearance level of the user; security roles that are used by a company to organize users in a hierarchical role structure according to the responsibilities of each type of work (each user can play more than one role); and security compartments that are used by an organization to classify users into a set of horizontal compartments or groups. In the ACA model an authorization subject is an identity composed of: userID (to identify the user), roleId (one or more user roles), compartmentID (one or more user compartments), securityLevel (a security level or a levels interval) and subjectExpression (OCL expression about the user profiles).



<<stereotype>> UserProfile
+comp 0..1
Compartment
+name: String
+secLev
SecurityLevel
+name: String
+supLevel
+infLevel
+role 0..*
Role
+name: String
+child 0..*
+root
Class
+ownedMember
Package
<<stereotype>> SecureClass
+attributes: Set(OclTypes)
+securityLevels: Levels
+securityRoles: Set(Role)
+securityCompartments: Set(Compartment)
<<stereotype>> SFact
<<stereotype>> SBase
+1..*
1..*
<<stereotype>> SDimension
+isTime
Constraint
<<stereotype>> AuditRule
+logType: AccessAttempt
<<stereotype>> AuthorizationRule
+ExceptSign: String = {+,-}
+ExceptPrivilege: Privilege
+involvedClasses: Set(OclType)
<<stereotype>> SecurityRule
+involvedClasses: Set(OclType)
<<stereotype>> SecurePackage
+securityLevel: Level
+securityRoles: Set(Role)
+securityCompartments: Set(Compartment)
<<stereotype>> SFactPackage
<<stereotype>> SDimensionPackage
<<stereotype>> SStarPackage
<<stereotype>> SecureInstance
+securityLevel: Level
+securityRoles: Set(Role)
+securityCompartments: Set(Compartment)
Instance
Property
<<stereotype>> SecureProperty
+securityLevels: Set(Level)
+securityRoles: Set(Role)
+securityCompartments: Set(Compartment)
<<stereotype>> SFA
+derivationRule: String
<<stereotype>> SDA
+derivationRule: String
<<stereotype>> SDD
<<stereotype>> SD
+derivationRule: String
<<stereotype>> SOID
<<stereotype>> SecureDW
+classes: Set(OclType)
+securityLevels: Sequence(Level)
+securityRoles: Role
+securityCompartments: Set(Compartment)
Model

**Fig. 2.** Secure Multidimensional PIM (SECDW).

According to MD models, we can identify the main authorization objects as follows: facts, dimension, classification, hierarchy levels, measures, dimension attributes and instances. The authorization object component is composed of two parts: an identity (that can be one of the following subattributes: class name or attribute name) and an objectExpression in OCL.

Our ACA model also allows us to define several kinds of security rules: Sensitive information assignment rules (SIAR) that specify multilevel security policies and allow us to define sensitivity information for each element in the MD model; Authorization rules (AUR) that specify the subject which the rule applies to, the object which the authorization refers to, the action which the rule refers to and the sign describing whether the rule permits or denies access; and Auditing rules (AR) that help us ensure that authorized users do not misuse their privileges.

SMD PIM metamodel (SECDW) is shown in Figure 2 and includes the main characteristics of Data Warehouses as many-to-many relations, degenerated dimensions, multiple classifications and the alternative path of hierarchies. We have improved this metamodel with several classes that allow us to represent our security classification in roles, levels and compartments in order to use them to carry out our transformations (i.e. for each security level we need to know what are the upper and inferior levels).

Security aspects can be defined according to our Access Control and Audit model. We can define security levels (SecurityLevels), user categories (SecurityCompartment), user roles (SecurityRoles) and security constraints (SConstraints) for each element of the metamodel: SecureFact, SecureDegenerateFact, SecureDimension, SecureBase, SecureDegenerateDimension, SecureFactAttribute, SecureDescriptor, SecureOID and SecureDimensionAttribute. Moreover, there is a UserProfile metaclass containing information about each users right of access to the multidimensional model.

According to our ACA model we can define security rules (SIAR), authorization rules (AUR) and audit rules (AU) by using OCL expressions and UML notes associated with the corresponding class.

## 2.2 Secure Multidimensional Code (SSAS Metamodel)

To implement security measures into OLAP tools we have selected SQL Server Analysis Services (SSAS) because it works with multidimensional models and allows us to define security measures over multidimensional elements (cube, dimension, cell). However, SSAS uses a role-based access control policy (RBAC) that is supposed to translate our measures defined according to our ACA model (with security roles, levels and compartments) into role approach.

Firstly, we have analyse source code in SSAS and we have defined the needed metamodels which represent multidimensional secure code with DW's structure and security measures in a previous step before final code that can be automatically obtained from these metamodels. SSAS defines a DW by using several XML source files. To obtain our SSAS metamodels we focus on roles configuration, DW's structure definition and how security constraints are specified.

In Figure 3 (a) is shown the role metamodel used to define roles in SSAS. We will use this metamodel to define roles in SSAS for each security role, level and compartment defined at conceptual level. Figure 3 (b) shows the cube metamodel to represent a cube in SSAS. We can define structural aspects for a cube as dimensions, attributes, hierarchies or measures, and security constraints by using permissions over cubes, dimensions or cells. Finally, in Figure 3 (c) we can see the dimension metamodel for defining dimensions and bases in SSAS by using Attributes and Hierarchies. Security

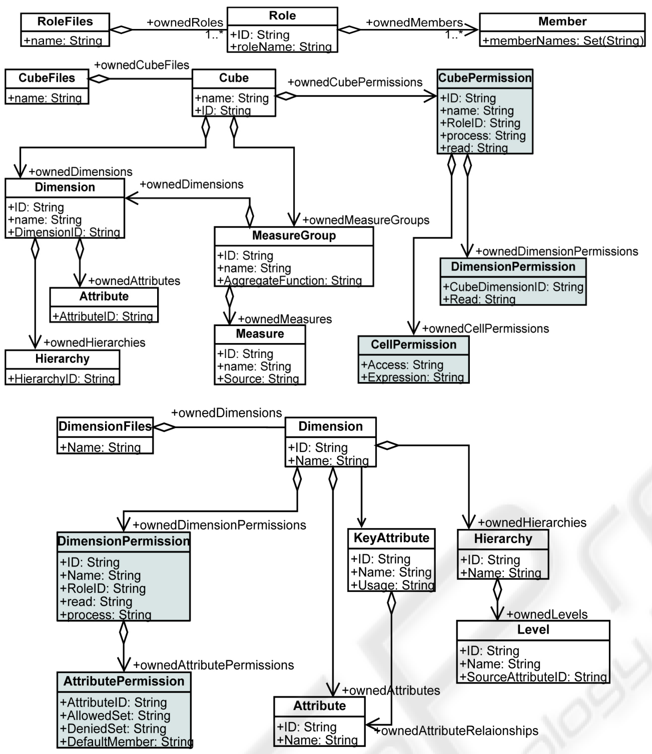


**Fig. 3.** SSAS Metamodel: Role configuration (a), Cube (b) and Dimension (c).

constraints can be established at dimension and cell levels and we can use complex MDX queries as allowed or denied set to define advanced security constraints.

## 3 Transformations

In order to obtain secure multidimensional code we follow the methodology previously defined in [10]. According to our ACA model we threat with levels, compartments and roles but in SSAS we use a role-based policy and in a first step we have to translate this security information by creating new roles for each possible classification.
SSAS uses an open policy with specific denials and we have to define what multidimensional elements are hidden for certain roles. Therefore, we have to analyse the security rules defined at conceptual level, to detect the involved roles and to hide them certain DW elements. To hide these elements we have to consider the concepts of security role, level and compartment. When access is denied to a certain security role this access has to be also denied to its descendants, and when is denied to a certain security level access has to be denied to each role that represents a lower security level.
Due to space constraints, this paper does not include the structural transformation and is focused on obtaining the secure multidimensional code corresponding to the security rules defined in our SECDW metamodel at conceptual level (see Figure 4). The analysis

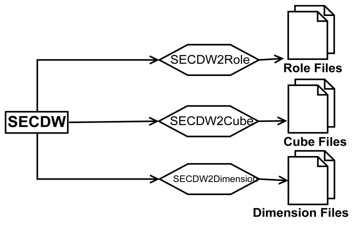


**Fig. 4.** General view of the proposed transformations.

and the automatic transformation of advanced security rules (SIAR and AUR) defined in UML notes with OCL expressions will be treated in future works. To define Audit rules in SSAS, administrators can directly establish audit activity on data including information about when data has been read and modified. These defined rules are presented in greater detail in Appendix and are composed of three main transformations (see Figure 4): SECDW2Role, SECDW2Cube and SECDW2Dimension. *S*ECDW2Role is in charge of generating the set of models representing the *"role files"*. After executing *SECDW2Role* the models produced contains enough information to easily produce the XML Role files. *S*ECDW2Cube obtains Cube files from the SECDW model. The *Cube files* are also a very important part in the DW description. Thus, the transformation *SECDW2Cube* tackles with the generation of models representing this files. This transformation deals on one hand with the generation of structure of the DW and on the other hand with the inclusion of the aforementioned security issues. Due to the lack of space, the structural rules has been omitted. Finally, *S*ECDW2Dimensions is in charge of producing the set of models representing the dimensions of the DW. Due to the extent of the transformation only the security issues has been taken into account for this paper.

## 4 Conclusions

We have accomplished an important step to complete our MDA architecture for developing secure Data Warehouses with the automatic generation of secure multidimensional code into an specific OLAP tool, SQL Server Analysis Services, by using QVT transformations.

In a previous work we obtained secure code for Oracle Label Security from a relational PSM metamodel, but we used a relational approach in a DBMS. In Data Warehouses, OLAP tools are more used than DBMS and we are focus our research effort on OLAP tools and multidimensional approaches.

Therefore, in this work we use the multidimensional capabilities of SSAS to translate the security measures defined in our secure multidimensional model at conceptual level into secure multidimensional code for SSAS. We obtain code with the role configuration of our system, the structural definition of the DW and the main part of the security measures defined at above levels. The final code adds other elements that can be easily obtained from our multidimensional secure code with an automatic transformation.

On the other hand, we realize that OLAP tools do not support the complete security requirements definition over the multidimensional model at upper abstraction levels and deal with partial security establishment. SSAS only uses RBAC as access control policy and our security model is richer than the security capabilities that SSAS offers. We have adapted our security information into a role-based approach and we have hidden multidimensional elements for certain roles to represents security rules. Furthermore, we can not represent security constraints on OLAP operations at upper abstraction levels allowing users to avoid the access to unauthorized information by using navigations or inferences.

Our MDA architecture for developing secure DW allows us to define security requirements but should be extended with the possibility of establishing navigations and inferences constraints that can be translated into OLAP code. In the same way, tools should be extended to give us control over navigations and inferences that can find out unauthorized enterprise information.

As a future work, we will define the remainder of the QVT transformations to obtain secure multidimensional code in SSAS from conceptual models. We will extend this work presenting the structural transformations that did not include due to space constraints. We will analyse the advanced security rules defined with OCL expressions and we will create the corresponding QVT transformations to complete our secure code. And finally, we will also extend this complete approach to obtain secure multidimensional code in other OLAP tools as Pentaho.

## Acknowledgements

This research is part of the ESFINGE (TIN2006-15175-C05-05) and METASIGN (TIN2004-00779) Projects financed by the Spanish Ministry of Education and Science, and of the MISTICO (PBC-06-0082) Project financed by the FEDER and the Regional Science and Technology Ministry of Castilla-La Mancha (Spain).

## References

1. Kimball, R.: The Data Warehouse Toolkit 2 Edition. John Wiley and Sons (2002)
2. Devanbu, P., Stubblebine, S.: Software engineering for security: a roadmap. ACM Press. Future of Software Engineering (2000) 227–239
3. Mouratidis, H., Giorgini, P.: An introduction. In: Integrating Security and Software Engineering: Advances and Future Visions. Idea Group Publishing (2006)
4. Denker, G., Kagal, L., Finin, T.: Security in the semantic web using owl. Information Security Technical Report 10 (2005) 51–58
5. Dhillon, G., Backhouse, J.: Information system security management in the new millennium. Communications of the ACM 43 (2000) 125–128
6. MDA, O.M.G.: Model driven architecture guide. (2003)
7. Czarnecki, K., Helsen, S.: Classification of model transformation approaches. (2003)
8. QVT, O.M.G.: (Omg mof 2.0 query, views, transformations request for proposals)
9. Fernández-Medina, E., Trujillo, J., Piattini, M.: Model driven multidimensional modeling of secure data warehouses. European Journal of Information Systems 16 (2007) 374–389

10. Blanco, C., Fernández-Medina, E., Trujillo, J., Piattini, M.: Implementing multidimensional security into olap tools. In: Third International Workshop ”Dependability Aspects on Data WArehousing and Mining applications” (DAWAM 2008), Barcelona, Spain, IEEE Computer Society (2008) 1248–1253
11. Soler, E., Stefanov, V., Mazón, J.N., Trujillo, J., Fernández-Medina, E., Piattini, M.: Towards comprehensive requirement analysis for data warehouses: Considering security requirements. In: Proccedings of The Third International Conference on Availability, Reliability and Security (ARES), Barcelona, Spain, IEEE Computer Society (2008) 104–111
12. Fernndez-Medina, E., Trujillo, J., Villarroel, R., Piattini, M.: Developing secure data warehouses with a uml extension. Information Systems 32 (2007) 826–856
13. Fernández-Medina, E., Trujillo, J., Villarroel, R., Piattini, M.: Access control and audit model for the multidimensional modeling of data warehouses. Decision Support Systems 42 (2006) 1270–1289
14. Soler, E., Villaroel, R., Trujillo, J., Fernndez-Medina, E., Piattini, M.: Representing security and audit rules for data warehouses at the logical level by using the common warehouse metamodel. In: 1st Int. Conference on Availability, Reliability and Security, Vienna, Austria (2006) 914–921
15. Soler, E., Trujillo, J., Fernndez-Medina, E., Piattini, M.: A set of qvt relations to transform pim to psm in the design of secure data warehouses. In: IEEE International Symposium on Frontiers on Availability, Reliability and Security (FARES 2007), Viena, Austria (2007) 644–651
16. Lujan-Mora, S., Trujillo, J., Song, I.Y.: A uml profile for multidimensional modeling in data warehouses. Data and Knowledge Engineering 59 (2006) 725–769

## Appendix

In this Appendix the QVT transformations defined to obtain secure multidimensional code from conceptual models defined according to our SECDW metamodel are presented focusing on security rules. The proposed transformations are divided into several relations, each of them is in charge of transforming elements from the source model into elements of the target model. It is possible to distinguish between two kinds of relations: relations and top relations. Top relations are mandatory and must always be held by both the source and target models. On the opposite, simple relations and only executed when other relations invokes them. Thus, top relation could be considered as a ”main” function or method.

**SECDW2Role**

```
transformation SECDW2Role(SECDW pim, Role psm){
top relation Package2RoleFiles {
  checkonly domain psm p:Package{
    name = n;
    ownedMember = OWNMEMB:Set(PackageableElement);   }
  enforce domain pim rf:RoleFiles{
    name = n;
    ownedRoles = OWNROLES:Set(Role);   }
  where{
    OWNMEMB->forAll(sr:SRole | OWNROLS->including
      (SRole2Role(sr)));
    OWNMEMB->forAll(sc:SCompartment |
      OWNROLS->including(SCompartment2Role(sc)));
    OWNMEMB->forAll(sl:SLevel | OWNROLS->including
      (SLevel2Role(sl)));   }
}
relation SCompartment2Role {
  checkonly domain psm sc:SCompartment{
    name = n;   }
  enforce domain pim r:Role{
    fileName = "SC"+n+".role";
    ID = "SC"+n;
    roleName = "SC"+n;
    ownedMembers = OWNEDMEMBS:Set(Member);   }
}
relation SRole2Role {
  checkonly domain psm sr:SRole{
    name = n;   }
  enforce domain pim r:Role{
    fileName = "SR"+n+".role";
    ID = "SR"+n;
    roleName = "SR"+n;
    ownedMembers = OWNMEMBS:Set(Member);   }
}
relation SLevel2Role {
  checkonly domain psm sl:SRole{
    name = n;   }
  enforce domain pim r:Role{
    fileName = "SL"+n+".role";
    ID = "SL"+n;
    roleName = "SL"+n;
    ownedMembers = OWNEDMEMBS:Set(Member);   }
}
```

**SECDW2Cube**

```
transformation SECDW2Cube (SECDW psm, Cube pim){
top relation Package2CubeFiles{
  checkonly domain psm p:Package{
    name = n;
    ownedMember = OWNMEMB:Set(PackageableElement); }
  enforce domaim pim d:DataWareHouse{
    name = n;
    ownedCubes = OWNCUBS:Set(Cube);   }
  where{
    OWNMEMB->forAll(sf:SFact |
      OWNCUBS->including(SFact2Cube(sf)));   }
}
relation SFact2Cube{
  checkonly domain psm sf:SFact{
    name = n;
    securityLevels= SECLEVS:Set(Level);
    securityRoles = SECROL:Set(Role);
    securityCompartments = SECCOMPS:Set(Compartment);
    ownedAttribute = OWNATTR:Set(Property);  }
  enforce domain pim c:Cube{
    name = n;
    ID = n;
    ownedCubePermissions = OWNCUBEPERMS:Set(CubePermission); }
  where{
    securityCompartments->forAll(sc:SCompartment |
      SCompartmentClass2CubePermission(sc,c));
    securityRoles->forAll(sr:SRole |
     functionGetRoleChilds(sl)->forAll(srChild:SRole|
     SRoleClass2CubePermission(sr,c));
    securityLevels->forAll(sl:SLevel |
     functionGetUpperLevels(sl)->forAll(slUpper:SLevel|
     SLevelClass2CubePermission(sl,c));   }
}
relation CreateMeasureGroups {…}
relation SProperty2Measure {…}
relation SDimension2Dimension {…}
relation ProcessSBase {…}
relation CreateOwnedHierarchies {…}
relation SProperty2Attribute {…}
relation SCompartmentClass2CubePermission{
  checkonly domain psm sc:SCompartment{
    name = n;   }
  enforce domain pim c:Cube{
    name = cubeName;
    ID = cubeName;
    ownedCubePermissions = OWNCUBEPERMS:Set(CubePermission);   }
  enforce domain pim cp:CubePermission{
    ID = "CubePermission"+n;
    name = "CubePermission"+n;
    RoleID = n;
    Process = "true";
    Read = "Allowed";   }
  where{
    OWNCUBEPERMS->including(cp);   }
}
relation SRoleClass2CubePermission{
  checkonly domain psm sr:SRole{
    name = n;   }
  enforce domain pim c:Cube{
    name = cubeName;
    ID = cubeName;
    ownedCubePermissions = OWNCUBEPERMS:Set(CubePermission);   }
  enforce domain pim cp:CubePermission{
    ID = "CubePermission"+n;
    name = "CubePermission"+n;
    RoleID = n;
    Process = "true";
    Read = "Allowed";   }
  where{
    OWNCUBEPERMS->including(cp);   }
}
relation SLevelClass2CubePermission{
  checkonly domain psm sl:SLevel{
    name = n;   }
  enforce domain pim c:Cube{
    name = cubeName;
    ID = cubeName;
    ownedCubePermissions = OWNCUBEPERMS:Set(CubePermission);  }
  enforce domain pim cp:CubePermission{
    ID = "CubePermission"+n;
    name = "CubePermission"+n;
    RoleID = n;
    Process = "true";
    Read = "Allowed";   }
  where{
    OWNCUBEPERMS->including(cp);   }
}
relation SCompartmentAtt2CellPermission {…}
relation SRoleAtt2CellPermission {…}
relation SLevelAtt2CellPermission {…}
```

**SECDW2Dimension**

```
transformation SECDW2Dimensions(SECDW pim, Dimensions psm){
top relation Package2DimensionFiles{
  checkonly domain pim p:Package{
    name = n;      ownedMembers   }
  enforce domain psm df:DimensionFiles{
    name = n;
    ownedDimensions = OWNDIMS:Set(Dimension);   }
  where{
    OWNMEMB->forAll(sd:SDimension | SDimension2Dimension(df,sd));   }
}
relation SDimension2Dimension{
  enforce domain psm df:DimensionFiles{
    name = n;   }
  checkonly domain pim sd:SDimension{
    name = sdName;
    securityLevels = SECLEVS:Set(SLevel);
    securityRoles = SECROLS:Set(SRole);
    securityCompartments = SECCOMPS:Set(SCompartment);
    ownedAttribute = OWNATTR:Set(Property);   }
  enforce domain psm d:Dimension{
    ID = sdName;      Name = sdName;   }
  when{   n = sd.owner.name;   }
  where{
    SDimension.owner.owner->forAll(sc:SCompartment |
      createDimensionSIARForSCompartment(sc,d));
      SECCOMPS->forAll(sc:SCompartment |
      authorizeSCompartment(sc,tempSCompartment));
    SDimension.owner.owner->forAll(sr:SRole |
      createDimensionSIARForSRole(sr,d));
      SECROLS->forAll(sr:SRole |
      getLeafSecurityRoles(sr)->forAll(tempSRole:SRole |
      authorizeSRole(d,tempSRole));
    SDimension.owner.owner->forAll(sl:SLevel |
      createDimensionSIARForSLevel(sl,d));
      SECLEVS->forAll(sl:SLevel |
      getUpperSecurityLevels(sl)->forAll(tempSLevel:SLevel |
      authorizeSLevel(d,tempSLevel));
    OWNATTR->select(oclIsKindOf(SecureProperty))->
      forAll(sp:SecureProperty | processSecureProperty(sd,d,sp))   }
}
relation KeyProperty2KeyAttribute {…}
relation NonKeyProperty2Attribute {…}
relation SBase2Attributes {…}
relation createDimensionSIARForSCompartment{
  checkonly domain pim sc:SCompartment{  name = compartmentName;   }
  enforce domain pim d:Dimension{
    name = dimName;
    ownedDimensionPermissions = OWNDIMPERMS:Set(DimensionPermission);   }
  enforce domain psm ap:DimensionPermission{
    ID = "DimensionPermission" + compartmentName;
    Name = "DimensionPermission" + compartmentName;
    RoleID = compartmentName;  }
}
relation createDimensionSIARForSRole{
  checkonly domain pim sr:SRole{  name = roleName;   }
  enforce domain pim d:Dimension{
    name = dimName;
    ownedDimensionPermissions = OWNDIMPERMS:Set(DimensionPermission);   }
  enforce domain psm ap:DimensionPermission{
    ID = "DimensionPermission" + roleName;
    Name = "DimensionPermission" + roleName;
    RoleID = roleName;   }
}
relation createDimensionSIARForSLevel{
  checkonly domain pim Sl:SLevel{  name = levelName;   }
  enforce domain pim d:Dimension{
    name = dimName;
    ownedDimensionPermissions = OWNDIMPERMS:Set(DimensionPermission);   }
  enforce domain psm ap:DimensionPermission{
    ID = "DimensionPermission" + levelName;
    Name = "DimensionPermission" + levelName;
    RoleID = levelName;   }
}
relation authorizeSCompartment {…}
relation authorizeSRole{
  enforce domain psm d:Dimension{
    name = n;
    ownedDimensionPermissions = OWNDIMPERMS:Set(DimensionPermission);   }
  checkonly domain pim sr:SRole{
    name = sRolName;   }
  where{
    let authDimPer:DimensionPermission = OWNDIMPERMS->
      select(ID = ("DimensionPermission"+sRolName)) in
      authDimPer.Read = "Allowed";         authDimPer.Process = "true";   }
}
relation authorizeSLevel{
  enforce domain psm d:Dimension{
    name = n;
    ownedDimensionPermissions = OWNDIMPERMS:Set(DimensionPermission);   }
  checkonly domain pim sl:SLevel{  name = slevelName;   }
  where{
    let authDimPer:DimensionPermission = OWNDIMPERMS->
      select(ID = ("DimensionPermission"+slevelName)) in
      authDimPer.Read = "Allowed";         authDimPer.Process = "true";   }
}
relation processSecureProperty{
  checkonly domain pim sd:SDimension{
    name = n;   }
  enforce domain psm d:Dimension{
    name = n;
    ownedDimensionPermissions = OWNDIMPER:Set(DimensionPermission);   }
  checkonly domain pim sp:SecureProperty{
    name = spName;
    securityLevels = SECLEVS:Set(SLevel);
    securityRoles = SECROLS:Set(SRole);
    securityCompartments = SECCOMPS:Set(SCompartment);   }
  where{
   SECLEVS->forAll(sl:SLevel |
     getUpperLevels(sl)->forAll(tmpSLevel:SLevel |
       createPositiveAttributePermisions(d.ownedDimensionPermisions->
       select(ID=("DimensionPermission"+tmpSLevel.name)),sp);
     getLowerLevels(sl)-> forAll(tmpSLevel:SLevel |
       createNegativeAttributePermisions(d.ownedDimensionPermisions->
       select(ID=("DimensionPermission"+tmpSLevel.name)),sp);
    SECROLS->forAll(sr:SRole |
      getLeafSRoles(sr)->forAll(tmpSRole:SRole |
        createPositiveAttributePermisions (d.ownedDimensionPermisions->
         select(ID=("DimensionPermission"+tmpSRole.name)),sp);
      getNonLeafRoles(sr)->forAll(tmpSRole:SRole |
        createNegativeAttributePermisions(d.ownedDimensionPermisions->
         select(ID=("DimensionPermission"+tmpSRole.name)),sp);
    SECLEVS->forAll(sc:SCompartment |
      createPositiveAttributePermisions (d.ownedDimensionPermisions->
      select(ID=("DimensionPermission"+sl.name)),sp);   }
}
relation createPositiveAttributePermisions{
  checkonly domain pim sp:SecureProperty{
    name = spName;   }
  enforce domain psm dp:DimensionPermission{
    ID = "DimensionPermission"+ID;
    Name = "DimensionPermission"+ID;
    ownedAttributePermissions = OWNATTPERMS:Set(AttributePermission);   }
  enforce domain psm at:AttributePermission{
    AttributeID = spName;   }
}
relation createNegativeAttributePermissions{
  checkonly domain pim sp:SecureProperty{
    name = spName;   }
  enforce domain psm dp:DimensionPermission{
    ID = "DimensionPermission"+ID;
    Name = "DimensionPermission"+ID;
    ownedAttributePermissions = OWNATTPERMS:Set(AttributePermission);   }
  enforce domain psm at:AttributePermission{
    AttributeID = spName;
    DeniedSet = "["+sp.class.name+"].["+sp.name+"]";   }
}
```