Automatic Generation of XACML Code using Model-Driven Approach

Athareh Fatemian, Bahman Zamani, Marzieh Masoumi, Mehran Kamranpour, Behrouz Tork Ladani, Shekoufeh Kolahdouz Rahimi
MDSE Research Group, Faculty of Computer Engineering
University of Isfahan
Isfahan, Iran
{fatemian_athareh, zamani, m.maasoumi, m.kamranpour, ladani, sh.rahimi}@eng.ui.ac.ir

***Abstract*- Precise specification of security requirements of software systems in general, and access control policies in particular, is a critical issue. The eXtensible Access Control Markup Language (XACML) is a well-known standard for defining access control policies. The problem is that using this language and manual formulation of policies requires technical knowledge and is error prone. To address this challenge, we propose a Domain-Specific Modeling Language (DSML), called Dual-XACML that supports both Role Based Access Control (RBAC) and Attribute Based Access Control (ABAC). As the tool support, a graphical editor as well as a transformation engine has been developed in this research. The graphical editor allows the user to create a model of access control policies for the target system. Then, using the transformations, the model is transformed into the corresponding XACML code. To evaluate the proposed approach, the XACML code of a typical system is generated, automatically**.

Keywords- Access Control Policies; XACML; ABAC; RBAC; MDE; DSML.

I. INTRODUCTION

In any information system, there exist several important security requirements, the protection of data and resources against unauthorized disclosure (i.e., confidentiality), preventing unauthorized or unwanted changes to data (i.e., integrity), ensuring access to data only by legitimate users, to name a few. To adhere to these requirements, any access to the system and the resources must be controlled, and only authorized access must be allowed. This process is called access control [1]. There are several ways to define access control policies, including eXtensible Access Control Markup Language (XACML), Next Generation Access Control (NGAC) [2], and Java ACL.

XACML is an XML-based language that determines who can do what in the system [3]. XACML allows us to define various access control policies governing a software system in the form of a set of policies, targets, and rules [4]. There exist different approaches for applying access control policies in XACML, including Attribute Based Access Control (ABAC), Role Based Access Control (RBAC), and Policy Based Access Control (PBAC) [5].

Manual formulation of access control policies using the XACML format is error prone, because of the possible changes in the policies governing a system, and the time-consuming process of implementation of changes. All these reasons, encourage the researchers for finding better approaches for generating XACML code.

One of the promising approaches that can help users in generating the codes is Model-Driven Engineering (MDE). Model-driven approaches, raise the level of abstraction by focusing on the models, and seek to automatically generate the corresponding code from the model [6]. This goal is achieved by defining a meta-model for the underlying domain (aka, Domain-Specific Modeling Language (DSML)) as well as building a transformation engine that is used for generating code from the models [6]. Using MDE techniques for domains such as access control policies, the focus of this paper, allows us to defining policies using models (higher abstraction), and then generate the XACML code, automatically.

There exist several literature work that used model-driven approaches in defining access control. Jin [7] used UML profile to overcome the complexity of XACML documents and to build an RBAC model, and facilitate the writing of RBAC policies in this language. The weak point of this work is that it only considered the RBAC and did not build a graphical editor. Busch et al. [8] used graphical modeling with UML-based notations to define security features, citing the difficulty of defining XACML policies in the XML language and the error prone nature of this process. To define the security aspects of web-based systems, Tout et al. [9] used a model-driven approach that extends Business Process Execution Language (BPEL) through the UML Profile. By providing a UML Profile, Tout et al. [10] covered all elements of XACML 3 and made it possible to design any policy in this language. These two works did not take advantage of a graphic editor with language-specific notations for their domain, and limited themselves to the default editor provided by the Eclipse. Nguyen et al. [11] proposed a model-driven security approach for access control management by formulating access control and delegation mechanisms that provide secure, flexible, and efficient access control. Poniszewska-Marańda [12] with the purpose to adopt concepts of MDA approach, access control, and especially role-

based access control, proposed a method for role-based access control in information systems in which RBAC and Usage Role Based Access Control (URBAC) are used for modeling.

In addition, there exist several research as well as commercial tools that have developed XACML editors. Nergaard et al. [13] created a graphical editor using the Scratch programming language, which helps build XACML policies based on the graphical elements of this programming language. Their tool resembles the state of XACML documents to the user. The problem is that to work with this tool, the user must be familiar with XACML syntax. Written in Java by the University of Morica in Spain, the UMU-XACML editor provides a user interface that allows you to define policies in a tree-like structure, but it is difficult to extend and modify this tool. The tool does not reduce the complexity of policy definition, nor does it specify how the policy works without reading the XACML code [13]. Stepian et al. [14] developed a simple policy editing language called Axiomatics Language for Authorization (ALFA) that can also be used to generate XACML policies. This editor does not have a sufficient level of abstraction. To summarize, Table I shows the comparison between the related works and the current research.

TABLE I. COMPARING THE RELATED WORKS

	High level of Abstraction	Supported by a Graphical Editor	Covering more than one type of XACML
Jin [7]	✓	-	-
Busch et al. [8]	✓	-	-
Tout et al. [9]	✓		-
Tout et al. [10]	✓	-	✓
Nguyen et al. [11]	✓	-	-
Poniszewska-Marańda [12]	✓	-	✓
Nergaard et al. [13]	-	✓	✓
Stepian et al. [14]	-	✓	✓
Present Study	✓	✓	✓

In the present study a DSML was designed that supports both RBAC and ABAC modes of the XACML. It allows the user to define both Role-Based Access Control and Attribute-Based Access Control policies through a single meta-model. The reason for choosing both ABAC and RBAC methods in designing our DSML is their popularity and widespread use [15]. These two methods are used to define access control policies in a number of recent studies in areas such as Blockchain technology [16], cloud computing security [17] and NoSQL database [18]. Due to the fact that the proposed meta-model allows the definition of access control policy with the two methods mentioned, it was named as Dual-XACML. To simplify the modeling process, a graphical editor was provided for the users through which they could create a model of their intended access control policies (one of the two modes ABAC or RBAC) for the target system and by running the defined transformation code, the user receives the corresponding XACML code. The superiority of this approach over the related work that were introduced before, is the simultaneous coverage of two widely used aspects of XACML, namely ABAC and RBAC. In addition, the accompanying graphical editor helps the user working in higher level of abstraction and get rid of the syntax of XACML.

To evaluate the usability of the proposed approach, a model was designed using the graphical editor and the corresponding XACML code was fully generated. The user can easily modify the model and generate the code automatically. This results in increased productivity as well as ease of use.

This article is organized as follows. Section II introduces basic information about access control policies and model-driven technology. Section III describes the steps of the proposed approach, which includes an overview of the approach and a description of its components. Section IV evaluates the proposed approach through a case study, and finally, Section V deals with conclusions and future work.

II. BACKGROUND

A. *Access Control concepts*

Access Control is the process by which a request is made to system resources and data, and decisions are made about granting access or rejecting a request. There are various policies to access control, which based on the different criteria governing the system, determine the do's and don'ts and authorized and unauthorized actions, and hence lead to security assurance [1]. Access control languages seek to articulate and implement policies. Due to the increasing number of applications that use XML as their data model, XML-based access control languages have been introduced. XACML, which is standardized by OASIS, is one of the most known languages in this category [19].

The RBAC approach is used for easy understanding of roles and assigning permissions to users based on their role in a system. A limited number of roles can cover all users of a system, and checking which users have access to what permissions can be done from the roles granted to them. Also, due to the comprehensibility of the roles, it is possible for non-specialist staff to assign roles to users. In ABAC, these are the attributes that are effective in determining the level of access of users. These attributes can be related to people or system resources and related to the environment and time of the access request. Determining the attributes in this approach should be done by experts [5].

B. *Model-Driven Engineering (MDE)*

In recent years, a new approach, known as Model-Driven Engineering (MDE), has been formed in software engineering in which the model as the main artifact plays a key role in the software development process and has applications beyond documentation [20]. The model by eliminating unnecessary details and reflecting the main features, increases the level of abstraction and understanding of a particular domain and it provides the possibility of producing sample systems for that

domain. In MDE, the main goal is to generate automated code from the model, which is done using transformations [6]. This is why the transformations are considered as the heart and soul of MDE [21].

III. THE PROPOSED APPROACH

As illustrated in Figure 1, this research includes three steps. Each step resulted in an artifact that can be used by the user who wants to generate the XACML code, automatically. In step 1, a DSML (i.e., a meta-model) called Dual-XACML, is designed and developed. In step 2, the model-to-text transformation engine is built. The transformations are saved in the *Generatesys.mtl* file. In step 3, a graphical editor is created to make it easier for the user to use the DSML and define access control policies. Each of these artifacts is described in the following.

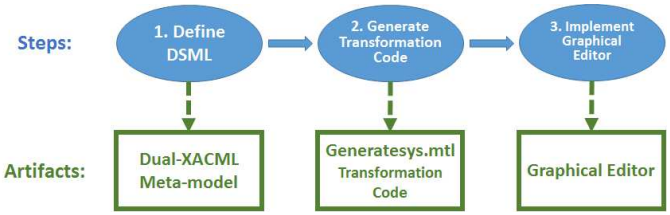

Fig. 1. The steps and artifacts of the proposed approach

A. Dual-XACML Meta-model

In order to apply access control with both role-based and attribute-based approaches, the Dual-XACML meta-model has been developed which supports both RBAC and ABAC policies. The meta-model is shown in Figure 2. With the aim of generating XACML code that supports both role-based and attribute-based policies through a single meta-model, only the basic elements required in the XACML language have been considered, and the details are eliminated. This meta-model, which includes 15 classes, is an adoption of the meta-models presented in previous research (i.e., [7], [12], [11], and [22]). We tried to preserve the main elements of RBAC and ABAC which are effective in generating XACML code, yet eliminating details and reducing complexity. In the following, the main concepts of our meta-model will be described.

PolicySet: In XACML, access control is defined as a set of policies that modeled by the **PolicySet** class in our meta-model. As the name implies, this class consists of a number of **Policy** classes. The policies in a **PolicySet** are executed through a policy combination algorithm specified in this class by the *policyCombineAlg* property, so that appropriate action can be taken based on the result of combining these policies

Policy: This class, which consists of the **Rule** and **Target** classes, represents the access control policy. Using the rule combining algorithm, which in this class is denoted by a property called *ruleCombineAlg*, the result of the rules that make up this policy are combined to determine the overall result of this policy. **Target** also indicates the purpose a policy.

Rule: This class represents the intended access control rule, which is evaluated separately. This rule alone is not decisive, and a set of those that make up a policy are influential in decision-making. This class itself consists of three components: **RuleEffect**, **Condition**, and **Target**. The **Condition** class specifies the conditional statement of the rule, and the **RuleEffect** class represents the effect of enforcing this conditional statement. The **Target** class also specifies the purpose of the rule.

Target: This class, which is itself a combination of the **Subject**, **Action**, and **Resource** classes, represents the goal that a policy pursues. That is, who does what action on which resource. The **Subject** class represents the entity that requested the access, the **Resource** class represents the resource on

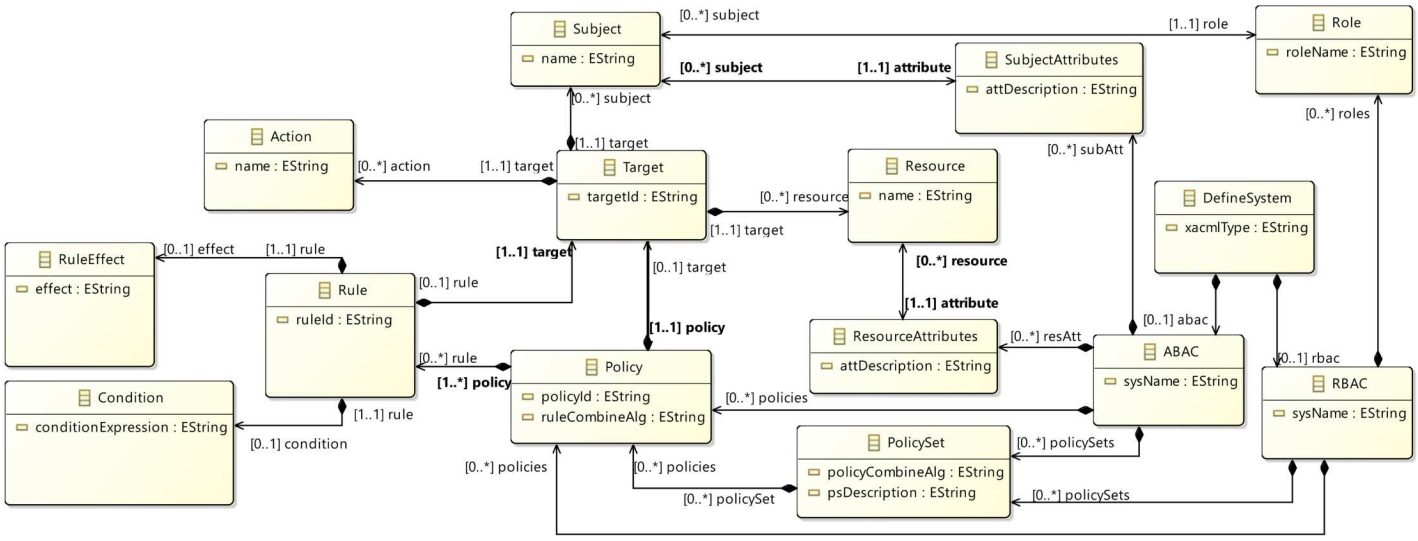

Fig. 2. Dual-XACML Meta-model

which the access request was issued, and the **Action** class indicates the action that the entity requested to take on the resource in request.

Role: This class represents the roles in the domain. Each **Subject** can have one **Role** and each **Role** can be assigned to multiple **Subject**s. This class is related to the RBAC approach.

SubjectAttribute: This class represents the attributes of the domain that each subject can have. Each **Subject** can have one or more **SubjectAttributes**, and each **SubjectAttribute** can be assigned to multiple **Subjects**. This class is related to the ABAC approach.

ResourceAttribute: This class represents the attributes of the domain that each resource can have. Each **Resource** can have multiple **ResourceAttributes**, and each **ResourceAttribute** can be assigned to multiple Resources. This class is related to the ABAC approach.

B. The generatesys.mtl transformation code

As mentioned before, automated code generation from the model is the most important goal in the MDE approach. In this research, to automatically generate XACML code from the corresponding model, The Acceleo[1] model-to-text transformation language is used. This language is a pattern-based technology that provides writing tools to create custom code generators for the user, so that the production of any source code from any data source in the form of EMF It is possible [23]. In the following, the transformation code written to automated generation of the XACML code, as well as sample output code will be described.

The main transformation file (called *generatesys.mtl*) navigates the classes in the source model, and based on the type of model (RBAC or ABAC), produces the corresponding XACML code. This transformation file contains 90 lines of code. Figure 3 shows part of this file that generates XACML code for the RBAC approach.

Lines 10 to 49 are written to generate the **PolicySet** code and its components. To do this, a *for* loop is used, which generates **PolicySet** tags for all **PolicySet**s related to the **RBAC** class, and between these two tags using a *for* loop (lines 12 to 48) inserts tags for all policies associated with each **PolicySet**. This *for* loop itself also contains loops for generating XACML code related to the **Target** and **Rule** associated with each **Policy**, and generates their tags and the values between them in the output. A *for* loop (lines 16 to 25) is also used to generate the **Subject**, **Resource**, and **Action** tags that make up a **Target**. The **Role** tag is generated through the connection between this class and the Subject class. A *for* loop (lines 28 to 46) is also used to generate the tags for each **Rule** and its components (**Target**, **Condition**, **RuleEffect**). Lines 52 to 88 in the transformation code to generate output code are related to Policies that are generated directly through the RBAC class based on the proposed meta-model and are not a subset of **PolicySets**. The transformation code of these policies is similar to the transformation code of policies that are members of **PolicySet** and is written in the same way. XACML code generation is done in the same way with the ABAC approach, except that instead of **Role**, **SubjectAttributes** related to the **Subject** are included in the code. Also **ResourceAttributes** associated with each **Resource** are also included in the code.

```
8  [for (aRBAC : RBAC | aDefineSystem.rbac.oclAsSet())]
9  <XACML Code For [aRBAC.sysName/] System Based On Role Based Access Control>
10 [for (ps : PolicySet | aRBAC.policysets.oclAsSet())]
11    <PolicySet PolicyDescription="[ps.psDescription/]"  PolicyCombineAlg= "[ps.policyCombineAlg/]">
12       [for (p1 : Policy | ps.policies.oclAsSet())]
13          <Policy PolicyId="[p1.policyId/]" RuleCombineAlg= "[p1.ruleCombineAlg/]">
14          [for (t1 : Target | p1.target.oclAsSet())]
15             <Target TargetId="[t1.targetId/]">
16             [for (s : Subject | t1.subject.oclAsSet())]
17                <Subject>  SubjectName="[s.name/]" </Subject>
18                <Role>  RoleName="[s.role.roleName/]" </Role>
19             [/for]
20             [for (a : Action | t1.action.oclAsSet())]
21                <Action>  ActionName="[a.name/]" </Action>
22             [/for]
23             [for (r : Resource | t1.resource.oclAsSet())]
24                <Resource>  ResourceName="[r.name/]" </Resource>
25             [/for]
26             </Target>
27       [/for]
28          [for (r1 : Rule | p1.rule.oclAsSet())]
29             <Rule RuleId="[r1.ruleId/]" RuleEffect="[r1.effect.effect/]">
30             [for (t2 : Target | r1.target.oclAsSet())]
31                <Target TargetId="[t2.targetId/]">
32                [for (s : Subject | t2.subject.oclAsSet())]
33                   <Subject>  SubjectName="[s.name/]" </Subject>
34                   <Role>  RoleName="[s.role.roleName/]" </Role>
35                [/for]
36                [for (a : Action | t2.action.oclAsSet())]
37                   <Action>  ActionName="[a.name/]" </Action>
38                [/for]
39                [for (r : Resource | t2.resource.oclAsSet())]
40                   <Resource>  ResourceName="[r.name/]" </Resource>
41                [/for]
42                </Target>
43       [/for]
44                <Condition>  Condition="[r1.condition.conditionExpression/]" </Condition>
45             </Rule>
46 [/for]
47          </Policy>
48 [/for]
49    </PolicySet>
50 [/for]
```

Fig. 3. Code excerpt of the *generatesys.mtl* transformation

C. Graphical Editor

To make it easy for the users, a graphical editor is built through which they can define the desired access control policies and get the corresponding XACML code as the output.

Sirius[2] tool, which is an Eclipse plugin, was used to create this graphical editor. Using Eclipse modeling technologies such as EMF, this tool makes it possible to create a graphic modeling workbench [24]. To implement the graphical editor, the meta-model must be associated with the graphical editor, and visual notation of each elements of the meta-model and relationships between them must be defined.

After assigning the notations to the meta-model elements, conditions must be provided so that the user can create the desired model by selecting the notation of each element and dragging and dropping it in the Ggraphical Editor. Figure 4 shows the modeling environment through this graphical editor. This graphical environment consists of two parts which are marked with numbers 1 and 2 in the figure: 1) design pane and 2) notation pane. The design pane is used to graphically design model and notation pane is used to select elements desired by the user. The user can define his/her model by dragging any element in the design pane and entering the properties of each element.

[1] https://www.eclipse.org/acceleo/

[2] https://www.eclipse.org/sirius/

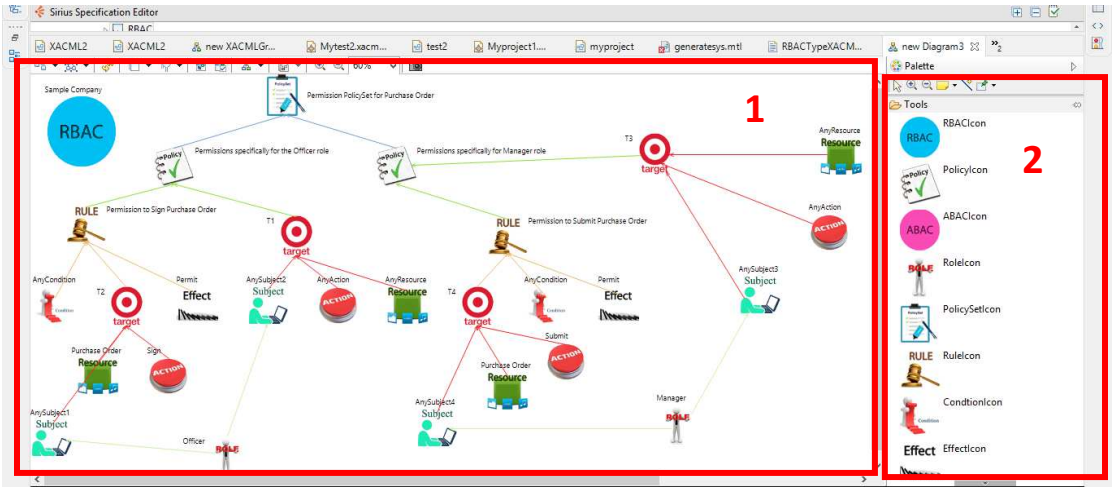

Fig. 4. View of the created graphical editor environment

IV. Case Study

To evaluate the proposed method and evaluate its power in generating XACML code corresponding to an input model, a model of RBAC for Procurement Department of Bam University of Medical Science was first constructed using a graphical editor, as shown in Figure 5. Then, by executing the described transformation code on the input model, the corresponding XACML code was generated. Figure 6 shows a part of the generated output code corresponding to the graphical input model. The sample model and its equivalent output code are a combination of the examples presented in [12] and [25].

After executing the transformation code, an output file was generated that is in XACML format. Due to the completeness of the transformation file, the code corresponding to the components of the model was generated completely and no information was lost during the transformation. The number of lines corresponds to size of input model. Obviously, the larger and more complex the model, the number of output file lines will be more, while in any case the number of transformation file lines is unchanged and no more code needs to be added. In addition, XACML output can be generated with both RBAC and ABAC approaches through the same code.

V. Conclusion

The model-driven engineering approach seeks to generate the desired code by creating DSML and using transformation engines to create diverse models from one language.

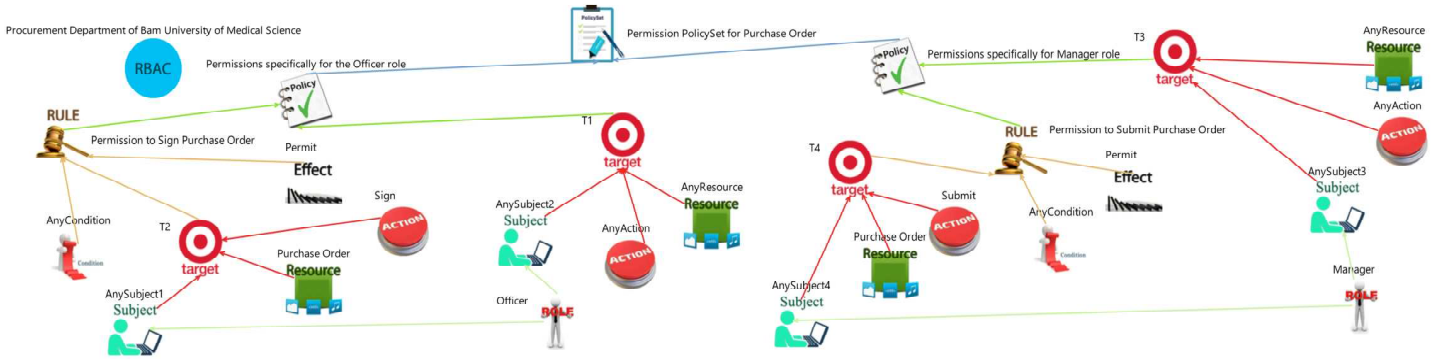

Fig. 5. Model designed using the graphical editor

In this research, by creating a Dual-XACML meta-model based on both types of roles and attributes and creating a model-to-text transformation code in Acceleo, it was possible for the user to generate XACML code through the model. The created meta-model has 15 classes and the written transformation code contains 90 lines of code. Also, by creating a graphical editor

```
<PolicySet PolicySetDescription="Permission PolicySet for Purchase Order"
                                        PolicyComAlg="permit-overrides">
   <Policy PolicyId="Permissions specifically for the Officer role"
                                    RuleCombAlg="permit-overrides">
     <Target  TargetId="T1">
       <Subject> SubjectName="AnySubject" </Subject>
       <Role> RoleName="Officer" </Role>
       <Resource> ResourceName="AnyResource" </Resource>
       <Actions> ActionName="AnyAction" </Actions>
    </Target>
    <Rule RuleId="Permission to Sign Purchase Order" RuleEffect="Permit">
      <Target TargetId="T2">
        <Subject> SubjectName="AnySubject" </Subject>
        <Resource> ResourceName="Purchase Order" </Resource>
        <Action> ActionName="Sign" </Action>
      </Target>
      <Condition> Condition="AnyCondition" </Condition>
    </Rule>
  </Policy>
  <Policy PolicyId="Permissions specifically for Manager role"
                          RuleCombiningAlg="permit-overrides">
    <Target TargetId="T3">
      <Subject> SubjectName="AnySubject" </Subject>
      <Role> RoleName="Manager" </Role>
      <Resource> ResourceName="AnyResource" </Resource>
      <Actions> ActionName="AnyAction" </Actions>
   </Target>
   <Rule RuleId="Permission to Submit Purchase Order" RuleEffect="Permit">
     <Target TargetId="T4">
        <Subject> SubjectName="AnySubject" </Subject>
        <Resource> ResourceName="Purchase Order" </Resource>
        <Action> ActionName="Submit" </Action>
     </Target>
```

Fig. 6. Part of the output code corresponding to the designed model

through the Sirius tool, it was possible for the user to create a model using understandable graphical notations. An important advantage of this approach is the simultaneous coverage of two widely used aspects of XACML through a single meta-model, as well as the design of a graphical editor with user-friendly notations that illuminates the MDE power to create code through the model, even without technical knowledge. Also, in this approach, only one transformation code was used to generate both RBAC and ABAC codes, which showed good strength and flexibility.

For future work, full coverage of XACML code can be created by extending the proposed meta-model to include all RBAC and ABAC elements. It is also possible to create a meta-model that can cover and produce other ways of defining access control policies. In addition, the power of the MDE in model-to-model transformations can be harnessed to enable mapping between XACML and other access control languages such as NGAC.

REFERENCES

[1] P. Samarati and S. De Capitani, "Access Control : Policies , Models , and," *Found. Secur. Anal. Des.*, pp. 137–196, 2001.

[2] D. F. Ferraiolo, R. Chandramouli, V. C. Hu, and D. R. R. Kuhn, "A Comparison of Attribute Based Access Control (ABAC) Standards for Data Service Applications," *NIST Spec. Publ. 800-178*, 2016, [Online]. Available: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-178.pdf.

[3] C. D. P. K. Ramli, H. R. Nielson, and F. Nielson, "The logic of XACML," *Sci. Comput. Program.*, vol. 83, pp. 80–105, 2014, doi: 10.1016/j.scico.2013.05.003.

[4] A. Anderson *et al.*, "Extensible Access Control Markup Language (Xacml) Version 1.0," *Oasis*, 2003.

[5] E. J. Coyne and T. R. Weil, "ABAC and RBAC: Scalable, flexible, and auditable access management," *IT Prof.*, vol. 15, no. 3, pp. 14–16, 2013, doi: 10.1109/MITP.2013.37.

[6] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Engineering in Practice Secend Edition.* Morgan & Calypool Publishers, 2017.

[7] X. Jin, "Applying Model Driven Architecture approach to Model Role Based Access Control System," 2006.

[8] M. Busch, N. Koch, M. Masi, R. Pugliese, and F. Tiezzi, "Towards model-driven development of access control policies for web applications," *Proc. Work. Model. Secur. MDEc 2012*, no. November 2016, 2012, doi: 10.1145/2422498.2422502.

[9] H. Tout, A. Mourad, H. Yahyaoui, C. Talhi, and H. Otrok, "Towards a BPEL model-driven approach for web services security," *2012 10th Annu. Int. Conf. Privacy, Secur. Trust. PST 2012*, pp. 120–127, 2012, doi: 10.1109/PST.2012.6297928.

[10] H. Tout and A. Mourad, "Model-Driven Specification and Design-Level Analysis of XACML Policies," in *ICNGCCT 2015*, 2015, no. April, doi: 10.13140/RG.2.1.2573.6167.

[11] P. H. Nguyen, G. Nain, J. Klein, T. Mouelhi, and Y. Le Traon, *Modularity and dynamic adaptation of flexibly secure systems: Model-driven adaptive delegation in access control management*, vol. 8400, no. April. 2014.

[12] A. Poniszewska-Marańda, "Model driven architecture for modeling of logical security based on RBAC approach," *J. Appl. Comput. Sci.*, vol. 22, no. 1, pp. 183–199, 2014.

[13] H. Nergaard, N. Ulltveit-Moe, and T. Gjøsæter, "A scratch-based graphical policy editor for XACML," *ICISSP 2015 - 1st Int. Conf. Inf. Syst. Secur. Privacy, Proc.*, pp. 182–190, 2015, doi: 10.5220/0005240101820190.

[14] B. Stepien, A. Felty, and S. Matwin, "A non-technical user-oriented display notation for XACML conditions," *Lect. Notes Bus. Inf. Process.*, vol. 26 LNBIP, pp. 54–64, 2009, doi: 10.1007/978-3-642-01187-0_5.

[15] S. Das, B. Mitra, V. Atluri, J. Vaidya, and S. Sural, "Policy Engineering in RBAC and ABAC," in *From Database to Cyber Security: Essays Dedicated to Sushil Jajodia on the Occasion of His 70th Birthday*, P. Samarati, I. Ray, and I. Ray, Eds. Cham: Springer International Publishing, 2018, pp. 24–54.

[16] X. Jiang, "A Blockchain-based Access Control Scheme," *J. Phys. Conf. Ser.*, vol. 1955, no. 1, 2021, doi: 10.1088/1742-6596/1955/1/012088.

[17] S. Alayda, N. A. Almowaysher, M. Humayun, and N. Z. Jhanjhi, "A novel hybrid approach for access control in cloud computing," *Int. J. Eng. Res. Technol.*, vol. 13, no. 11, pp. 3404–3414, 2020, doi: 10.37624/ijert/13.11.2020.3404-3414.

[18] E. Gupta, S. Sural, J. Vaidya, and V. Atluri, "Attribute-Based Access Control for NoSQL Databases," *CODASPY 2021 - Proc. 11th ACM Conf. Data Appl. Secur. Priv.*, pp. 317–319, 2021, doi: 10.1145/3422337.3450323.

[19] S. De Capitani Di Vimercati, P. Samarati, and S. Jajodia, "Policies, models, and languages for access control," *Lect. Notes Comput. Sci.*, vol. 3433, pp. 225–237, 2005, doi: 10.1007/978-3-540-31970-2_18.

[20] A. Rodrigues Da Silva, "Model-driven engineering: A survey supported by the unified conceptual model," *Comput. Lang. Syst. Struct.*, vol. 43, no. Model Driven Introduction, pp. 139–155, 2015, doi: 10.1016/j.cl.2015.06.001.

[21] S. Sendall and W. Kozaczynski, "Model transformation: The heart and soul of model-driven software development," *IEEE Softw.*, vol. 20, no. 5, pp. 42–45, 2003, doi: 10.1109/MS.2003.1231150.

[22] D. Sivalingapandi, "Comparison and Alignment of Access Control Models," 2017.

[23] "Generate anything from any EMF model." https://www.eclipse.org/acceleo (accessed May 01, 2021).

[24] "What is Sirius?" https://www.eclipse.org/sirius/overview.html (accessed Jun. 15, 2021).

[25] A. Anderson, "XACML RBAC Profile." https://www.oasis-open.org/committees/download.php/2244/XACML_RBAC_Profile.html (accessed Apr. 06, 2021).