Collaboration and the Knowledge Economy: Issues, Applications, Case Studies
Paul Cunningham and Miriam Cunningham (Eds)
IOS Press, 2008 Amsterdam
ISBN 978–1–58603–924-0

A Meta Model Generator for Implementing Access Control and Security Policies in Distributed Systems Based on Model-Driven Architecture

Mirad ZADIC, Andrea NOWAK
Austrian Research Centers GmbH-ARC, A-2444 Seibersdorf, Austria
Tel: +43 50550-3155, Fax: +43 50550-2813, Email: {firstname.lastname}@acrs.ac.at

Abstract: Specifying security policies, especially access control policies, may be an enormous task for every security administrator. With the current vast and complex IT-infrastructure having sensitive assets, it is preferable to have a tool that simplifies the security policy specification task. Our goal in this work is to present the concept and design of a tool, which should solve the previous problem. To facilitate managing and maintaining security policies we are going to define a specific modeling language for generating tool on the base the features of Meta Modeling Environment. This provides a set of the visualization components and code generation capabilities to build layers of security policies architecture. The policy generator generates the access control policies in the XACML (eXtensible Access Control Mark-up Language) format.
Model-Driven Development (MDD) is a popular approach to creating solutions in a large number of domains. In many cases, however, an organization does not have the resources or time available to develop a graphical modeling environment from scratch using the Eclipse Modeling Framework (EMF), Graphical Editor Framework (GEF), or Graphical Modeling Framework (GMF). A way to reduce complexity of developing a Graphical Modeling and Generating Tool is to examine the Generic Eclipse Modeling System (GEMS), which makes possible rapidly creating such modeling tool from a visual language description or metamodel without any coding in third-generation languages.

Keywords: Model Driven Development, Security policies Architectures, Meta modeling, Domain-specific Modeling Languages, Policy Generator.

1. Introduction

The distributed service architectures without appropriate security guards carry indefinitely risks and for many companies is the security one of the biggest barriers for the realization of linking applications within organizations, across enterprises or across the Internet. If a continuous confidentiality, integrity, non-repudiation and access control is not guaranteed, the use of Web services is in the e-business doomed to failure.

1.1 Fali Recenica

A Web service end-point Security involves a number of aspects, such as: reliable messages, privacy, authorization, trust, authentication and cryptographic security. Each aspect addresses a number of optional features and parameters, which must be coordinated between communicating end-points. Currently, in most companies, the specification policies are manually created and managed by a security administrator as an independent procedure during the deployment stage after the software design and development. Because XACML policies [2] [3] are rich XML-based documents with complex syntax, it is very

difficult and time consuming to write error free XML documents by hand, especially for people who do not know the syntax well. This may affect the productivity of the administrator and the quality of the software. Hence, a suitable XACML editor or a tool that can automatically generate XACML policies is needed. Another problem is normally the administrator does not take part directly in the application design process, so he/she may not understand the software structure and details well enough to define policies to fulfil the security requirements.

Many models have been developed to construct and manage the security requirements but our security infrastructure is deployed in a modular way with pre-defined, configurable security components. In this paper we present the a modeling tool and generator which uses the Domain-Specific Modeling Languages (DSMLs), Model-Driven Engineering (MDE), Model-Driven Architectures (MDA), and Model Driven Development (MDD) [6]. The goal in this work is to present the concept and design of a generator, which should support the process of specifying the access control policies. Furthermore, the generator should provide the security administrator with expert knowledge and help him to tackle the well-known problems, which may arise during the security policy specification task.

In Section 1 and 2 we provide some background on the standards and the technologies our work is based and our main research objectives. Our presented Model-Driven-Approach with the metamodel, and there transformations tool for policy specifications are discussed in Section 3 and 4. Section 5 presents a short presentation of results. Section 7 we describe our future work and at least Section 8 gives an overview about related work

2. Objectives

Systems for model-driven software development can be seen as a new generation of visual programming languages. A system can be modeled at different levels of abstraction or from different perspectives. The syntax of every model is defined by a metamodel that's mean the metamodel defines the syntax of the modeling languages, a model plays the role of the source code, and the generator replaces the compiler.

Using this approach, it is possible to generate automatically large amounts of source code and other artifacts, e.g. deployment descriptors and make files, based on relatively concise models. This improves the productivity of the development process and quality of the resulting systems. It is also a step towards platform independent design of systems.

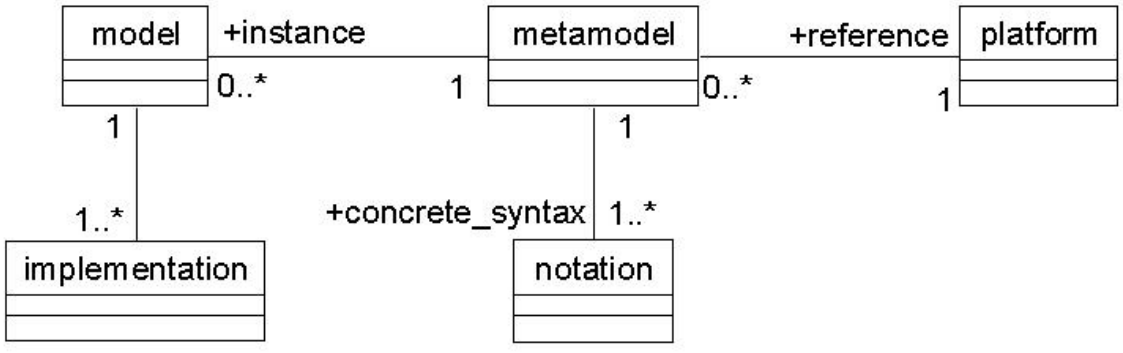

Figure 1: Different Levels of Modeling

The types and relationships have the following meaning: A model represents a software system at an appropriate level of abstraction or from a certain perspective. One or more implementations are generated from a model. A metamodel defines the syntax of a class of models. Every metamodel refers to a particular platform, called its reference platform (reference). A platform is an execution environment for software systems, like the Java

platform. The semantics of a metamodel is defined through transformation rules that map every language construct to constructs in the reference platform, Figure 1.

The Generic Eclipse Modeling System (GEMS), a part of the Eclipse Generative Modeling Technologies (GMT) project, helps developers rapidly create a graphical modeling tool from a visual language description (metamodel) without any coding in third-generation languages. GEMS substantially reduces the cost of developing a graphical modeling tool by allowing developers to focus on the key aspects of their tool: the specification of the DSML and the intellectual assets built around the use of the language. The infrastructure to create, edit, and constrain instances of the language is generated automatically by GEMS from the language specification. Moreover, GEMS allows the separation of language development, coding, as well as "look and feel" development, such as changing how modeling elements appear based on domain analyses.

3. Methodology

3.1 XACML Policies

The policy generator generates the access control policies in XACML (eXtensible Access Control Mark-up Language) format, which is established by OASIS. The XACML standard defines the machine-interpretable and also machine-enforceable security policies [2] [3] [4] [5]. Whenever a principal requests access to a resource, that request is passed to a software component called a Policy Decision Point (PDP). A PDP evaluates the request against the specified access control policies, and permits or denies the request accordingly. Actually the interface of the whole environment to the outside world is Policy Enforcement Point (PEP). It receives the access requests and evaluates them with the help of the other actors and permits or denies the access to the resource. The PEP forms a request based on the requester's attributes, the resource in question, the action, and other information pertaining to the request. The PEP sends then this request to a PDP.

The main goal of policy generator [7] [8] is to derive low-level and enforceable security policies. These policies are deduced from the business process, which acts as the input scenario. In this scenario, the security problems should be already identified, and annotated within the business process. The generator shall generate follows types of XACML components, Figure 2:

<PolicySet> element combines Policies in a PolicySet. It has a target, a policy-combining algorithm-identifier, a set of policies and obligations.

<Policy> element combines rules in a policy. Therefore, a policy consists of a target element, a rule-combining algorithm identifier, a set of rules and obligations. The target element of policy has the same purpose as target element of rule. The rule-combining algorithm determines rule-combining method applied to the rules.

<Rule> element is the most elementary unit of XACML policy. It has the target element, the condition element and the effect element. The target element specifies the environment, on which this rule should apply. The condition element is a Boolean function over subject, resource action and environment attributes. In order to apply the effect of rule on the target, this condition should evaluate to true, otherwise, the rule will return indeterminate. Lastly, the effect element states the decision for this rule: either permit or deny.

<Target> element describes the properties of the environment, on which the Rule, Policy or Policy Set should apply. It contains the subject description, resource description, action performed and environment description.

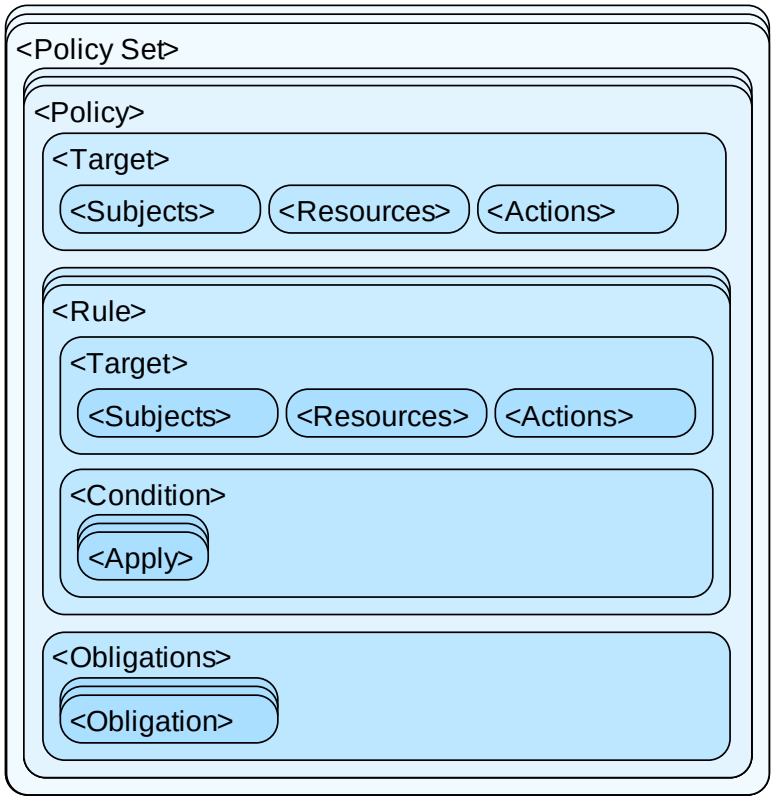

Figure 2: XACML Components

<Condition> element represents a Boolean expression that refines the applicability of the rule beyond the predicates implied by its target.

<Obligations> element of an XACML <Policy> is a directive to the PEP to perform additional processing following the enforcement of an access control decision. It contains one or more <Obligation> elements and typically references elements in the request context. Processing of <Obligations> elements is application-specific.

3.2 Meta Modeling Tool

The Generic Eclipse Modeling System (GEMS) created by the Distributed Object Computing (DOC) Group at the Institute for Software Integrated Systems (ISIS) at Vanderbilt University is an Eclipse-based Meta-configurable Modeling Environment that enables developers to generate Domain-Specific Modeling Language (DSML) tools for customizing software architecture via the following capabilities [9] [10] [11] [12]:

1. A visual interface that supports the creation of do-main-specific modeling languages (DSMLs), i.e., GEMS contains a metamodeling environment that supports the definition of paradigms, which are type systems that describe the roles and relationships in particular domains.
2. The creation of models that are instances of DSML paradigms within the same environment.
3. Customization of such environments so that the elements of the modeling language represent the elements of the domain in a much more intuitive manner than is possible via third-generation programming languages.
4. A flexible type system that allows inheritance and instantiation of elements of modeling languages.
5. An integrated constraint definition and enforcement module that can be used to define rules to be adhered to by elements of the models built using a particular DSML.
6. Facilities to plug-in analysis and synthesis tools that operate on the models.

GEMS provides a mechanism called Model Intelligence Guides (MIGs) to substantially reduce the cost of integrating a constraint solver and reduce the complexity of modeling large and complex domains.

4. Technology Description

4.1 XACML Policies Model Description

This section gives a short overview of our model driven development process for developing a XACML policy specification model. The presented XACML policy model defines end-point policies for Web Services. All model components make a base of challenges that should be generated from the policy generator. XACML components are reflected in the relevant structure of WSDL:

Port → Operation → Message

The established XCAML model has a set of definitions to facilitate combining of end-policies for service-oriented structure.

That means an XACML <PolicySet> element is associated with a concrete Web-service end-point definition. It is usually that its <Target> element has to identify Web-services port and to describe port's parameters. In the case that a policy must be target more finely than a port, as in presented XACML model, the <PolicySet> contains more <Policy> elements. The <Policy> elements define the objectives for each aspect of policy associated with the port. The policy-combining algorithm for <PolicySet> is defined with:

"urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides".

The content of <PolicySet/Target/Subjects> has only one <Subject> element for service-requestor. The MatchId for the <PolicySet/Target/ Subjects/Subject> element is defined with:

"urn:oasis:names:tc:xacml:1.0:function:string-equal"

<AttributeValue> and <SubjectAttributeDesignator> ere defined for a service-requestor.

The content of the <PolicySet/Target/Resources> element is the name attribute of the end-point's port definition. The MatchId for the <PolicySet/Target/Resources> element is defined with:

"urn:oasis:names:tc:xacml:1.0:function:anyURI-equal".

The contents of <PolicySet/Target/Actions> element is defined as <AnyAction/>.

There are two <Policy> elements one for service-requestor and one for service-provider. Therefore each <Policy> element is associated with a single aspect of an end-point policy concept, respectively "sending" and "receiving". The rule-combining algorithm for a <Policy> element is predefined with:

"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides".

The <Target> element of a <Policy> element "sending" has to identify the one objective of end-point policy to which to be successfully invoked. The content of <Policy/Target/Actions> element is defined with one <Action> element, which defined invoked object. The MatchId for the <Policy/Target/Actions/Action> is:

"urn:oasis:names:tc:xacml:1.0:function:anyURI-equal"

In <AttributeValue> is defined invoke for an object of Web-service port.

This "sending" <Policy> element has <Rule> element, which define strategy and it is affected as "Permit". In this case, the policy specifies the actions that must be performed,

either instead of, or in addition to, actions that may be performed. This facility for action defines the conjunction with policy evaluation in the <Obligations> element. The <Rule> strategy “Permit” is resulted unless PEP understand and discharge all of the <Obligations> elements associated with the applicable policy.

The <Obligations> element is data-processing obligation and annotates defined action operation. The FulfillOn attribute indicates the effect for which this obligation must be fulfilled by the PEP, defined as “Permit”. ObligationId (of type URI) is mapped to a specific handler called by the PEP and Obligation parameter values are passed to handler.

<Policy> element “receiving” in presented model carried the <Policy/Target/Subjects> and the <Policy/Target/Resources> elements but they are defined as <AnySubject/> and <AnyResource/>, respectively. In this case should be the contents of the <Policy/Target/Actions> element identify the invocation of Web-service object. The <AttributeValue> and <ActionAttributeDesignator>elements describe the invocation of Web-service object in the <Action> element. The <ActionMatch> element carries MatchId:

"urn:oasis:names:tc:xacml:1.0:function:anyURI-equal".

The <Rule> element contains set of <Apply> elements that define what to do in that invocation. The <Apply> elements are structured as follows:

```
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
        <AttributeSelector RequestContextPath="…" DataType="…"/>
    </Apply>
      <AttributeValue DataType="…"> … </AttributeValue>
</Apply>
```

There are three possible attribute’s classes: Unconstrained attributes, Constrained attributes and Authorized attributes. This XACML model preferred a constrained (environmental) attribute, whose value is outside the control of the policy-user. The emergency condition code is an example of an environmental attribute over which a policy-user has no control. In presented model this attribute is used to evaluate any action that the policy-user might take, for instance is mapped to a specific handler.

```
<AttributeSelector RequestContextPath="//Environment/Attribute
                    [...handler...]/AttributeValue/text()” DataType=”…”/>
```

There is a subject attribute over handler action in the elements of message. In this case of the environmental attribute a “handler“ takes care about signing or encrypting of elements in SOAP messages. The policy-user waits until the predicate involving the result. The <Rule/Target> element is omitted.

4.2 Meta Modeling Environment

The Generic Eclipse Modeling System (GEMS) as part of the Eclipse Generative Modeling Technologies (GMT) project is used for design and implement our generator. GEMS allows domain experts to rapidly create complex graphical modeling tools for Eclipse without writing Graphical User Interface (GUI), EMF, or XML code. GEMS provides extensive support for integrating intelligent mechanisms into a modeling tool to provide visual modeling queues, constraint-compliant batch processing, simulation, and analysis.

The implementation of GEMS reuses the basic Eclipse components such as EMF (Eclipse Modeling Framework) and GEF (Graphical Editing Framework), as well as parts of GMF (Graphical Modeling Framework) runtime. GMF utilizes Eclipse EMF and GEF technologies. EMF is used for model management and GEF for graphical user interface. GMF uses a static-mapping-based approach. It defines a set of meta-models: graphical

(presentation), tooling and mapping meta-models. Graphical meta-model defines the graphical element types. Tooling meta-model defines the palette and menus. The mapping meta-model defines the mapping possibilities between the models. In addition, it uses EMF Ecore Model as the domain meta-model.

The main distinguishing feature of GEMS is an appropriately built presentation meta-model. It enables a clear separation of responsibilities between the GEMS presentation engine, which handles all the low-level presentation and layout-related tasks, and transformations, which create and maintain only the domain and the logical structure of presentation. GEMS contains also a universal metamodel-based presentation engine for element property editing, and an advanced project tree engine.

GEMS provides an extensive feature set for rapidly building graphical modeling tools including [13] [14] [15]:

1. A graphical language for metamodel specification that can capture
   - Domain entities
   - Attributes of entities
   - Inheritance relationships
   - Connection and containment relationships between entities
   - Distinct modeling views
   - Graphical information, such as connection styles, colors, and fonts
   - Constraints
2. A code generation framework, which does not require any coding or XML editing, for transforming a GEMS metamodel into a working Draw2D/GEF Eclipse plug-in for editing instances of the language
3. The visual appearance of the generated modeling tool can be customized by creating CSS stylesheets to modify the icons, colors, fonts, connection styles, and other visual attributes of the modeling entities
4. The views available to a modeler can be customized through a mechanism similar to CSS stylesheets
5. The generated graphical modeling plug-ins, created by GEMS, support extensive external customization through extension points for
   - Adding code generators and transformation, such as Open Architecture Ware, Java Emitter Templates, and Atlas Transformation Language
   - Model pre and post processing
   - Model event listeners
   - Triggers for invoking actions (similar to database triggers)
   - Constraint languages
   - Menus
   - Palette customizers
   - Intelligent modeling guides
   - Model serializes
6. GEMS provides built-in support for constraints written in Java, OCL, and Prolog
   - Constraints can also be used as triggers for invoking actions

5. Developments

This section represents how to generate the XACML Policy Specifications modeling tool based on the metamodel describing syntax of the DSML (GEMS). As a case study, is described the development of a tool, called ARCSecuModel, for specifying and modeling the security policies for access to the Web services. ARCSecuModel allows IT professionals to describe the needed policies component they wish to deploy security requirements of the Web-service and the security goals in the predefined Business processes.

For modeling the presented ARCSecuMetaModel is used GEMS to visually describe a metamodel in Eclipse, to generate the EMF, Draw2D, and GEF code and to implement a modeling tool for the language described by the metamodel. The Meta model is described with follow GEMS objects: Class, Connection, Attribute and Inheritance.

The first entity (class) is the ArcSecuModel, which is represented as the main white canvas containing the Domain and Messages. The ArcSecuModel is the root entity in our metamodel. Domain and Messages entities can be seen on the left and below from ArcSecuModel, respectively. Tree more entity types are present for security goals: IntegrityRequirement, ConfRequirement and NonRepRequirement.

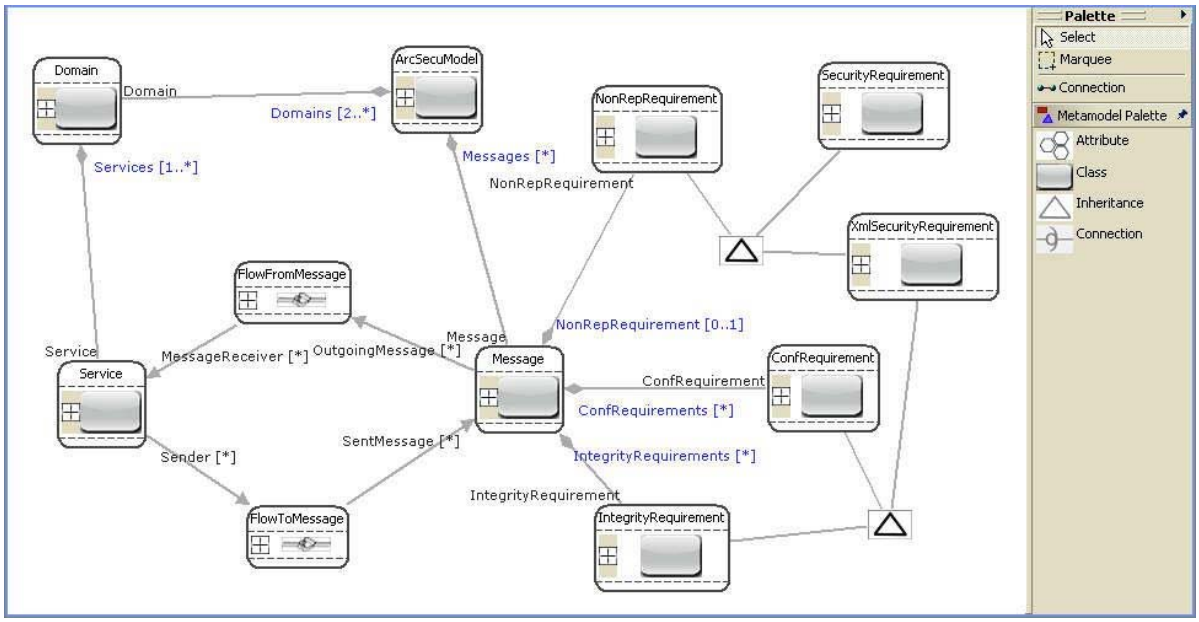

Figure 3: Metamodel for Web-Services Security Requirements

The security goals (IntegrityRequirement, ConfRequirement, NonRepRequirement) are child entities of Message. The Service entity is child entity from Domain. The finishing of ARCSecuMetaModel tool involved also connections between Message and Service. These are defined with FlowFromMessage and FlowToMessage connections that were defined in suitable directions.

Inheritance relationships are created by adding an Inheritance element to the model, and connecting it to the parent type and derived type(s). Derived types inherit any attributes, connections, or containment relationships specified by the parent. The presented ARCSecuMetaModel contains entities with two Inheritance relationships: first Inheritance between IntegrityRequirement (BaseClass), ConfRequirement (BaseClass), XmlSecurityRequirement (DerivedFrom) and the second between NonRepRequirement (BaseClass), XmlSecurityRequirement (BaseClass), SecurityRequirement (DerivedFrom).

The classes IntegrityRequirement and ConfRequirement Attributes inherit the attributes from XmlSecurityRequirement entity (NS1; NS2; NS3; NS4; NS5; NS6; NS7; XPath). Service carries an attribute (string type) named OperationName and Massege carries also an attribute (Boolean type) named IsResponse. Figure 3 shows the complete metamodel ARCSecuMetaModel for our policy generator tool.

After creating a metamodel, GEMS's DSML plug-in generator creates the modeling tool. The code generator first traverses the various entities in the model and generates an Ecore model, which is a set of EMF objects that represent the metamodel or syntax of the visual language. GEMS then invokes the appropriate EMF code generators to produce EMF classes that implement the Ecore definition.

The generated EMF classes are used as the object graph underlying ARCSecuModel. These classes automate key aspects of serialization and de-serialization. By default, GEMS leverages EMF's ability to save EMF models to XMI. Other serializers can be plugged into persist models in a database or use an alternate file format. The EMF object graph also allows GEMS to leverage the libraries available for Eclipse to check Object Constraint Language (OCL) constraints against a model.

The second set of code generated by GEMS, are the classes required to plug the generated EMF code into the GEMS runtime framework. The GEMS runtime provides a layer built on top of GEF (and soon GMF) that provides higher level capabilities, such as applying CSS styles to elements, exposing remote update mechanisms, and providing constraint solver modeling intelligence.

The GEMS runtime provides numerous extension points for adding custom functionality to the generated modeling tool. Extension points are available for adding actions that can be triggered by OCL, Java or role-based object constraint assertions on the model, customization of menus, custom code generators, remoting mechanisms, custom serializers, and many other features.

The third set of code artifacts, generated by GEMS, are the various XML descriptors, build specifications, class path directives, and icons required to integrate the generated tool into Eclipse as a plug-in. When a modeling tool is generated into a Java project, these various artifacts configure the project properly to build the plug-in and make it visible for testing with the runtime workbench. The build artifacts also configure the project so that it can be exported properly as an Eclipse plug-in modeling tool.

One benefit of GEMS is to avoid requiring developers to use third-generation languages to write any graphics code to implement a modeling tool. GEMS allows developers to customize the look and feel of their modeling tools. It supports this capability by allowing developers to change the default icons visible on the palette and in the model, which enables developers to quickly create impressive visualizations matching domain notations.

Swapping icons isn't the only mechanism GEMS provides to customize the look and feel of modeling tools like ARCSecuModel. Developers can use CSS style sheets to change fonts, colors, backgrounds, background images, line styles, and many more features of a GEMS-based modeling tool as shows the Figure 4.

In GEMS CSS styles are applied to elements using traditional CSS selectors. The key difference is that selectors refer to the roles and types specified in the metamodel. Even more complex visual behaviors can be created by leveraging a feature called TAGS in GEMS. TAGS are textual markup that can be added to modeling elements to denote that they have a certain property. The GEMS TAGS facility supports the combination of domain analysis with CSS styles to rapidly develop complex domain-specific visual behaviors, such as changing the icon for a Domain that doesn't have sufficient resources (Services).

Triggers allow constraints to perform a number of actions to guarantee model correctness, such as showing a warning message, vetoing a model change, or adding elements to fix the error. A trigger can also result in a message popping up above the modeling element that has violated the constraint. Much more complex actions can take place as well. For example, a constraint violation can trigger an action that runs policy generating process.

6. Results

The modeling tool is an Eclipse rich client that is used to visually model the security needs of each communication and generate the XACML policies for each involved domain. Figure 4 shows some instances of a security model that describes the secure communication between domains.

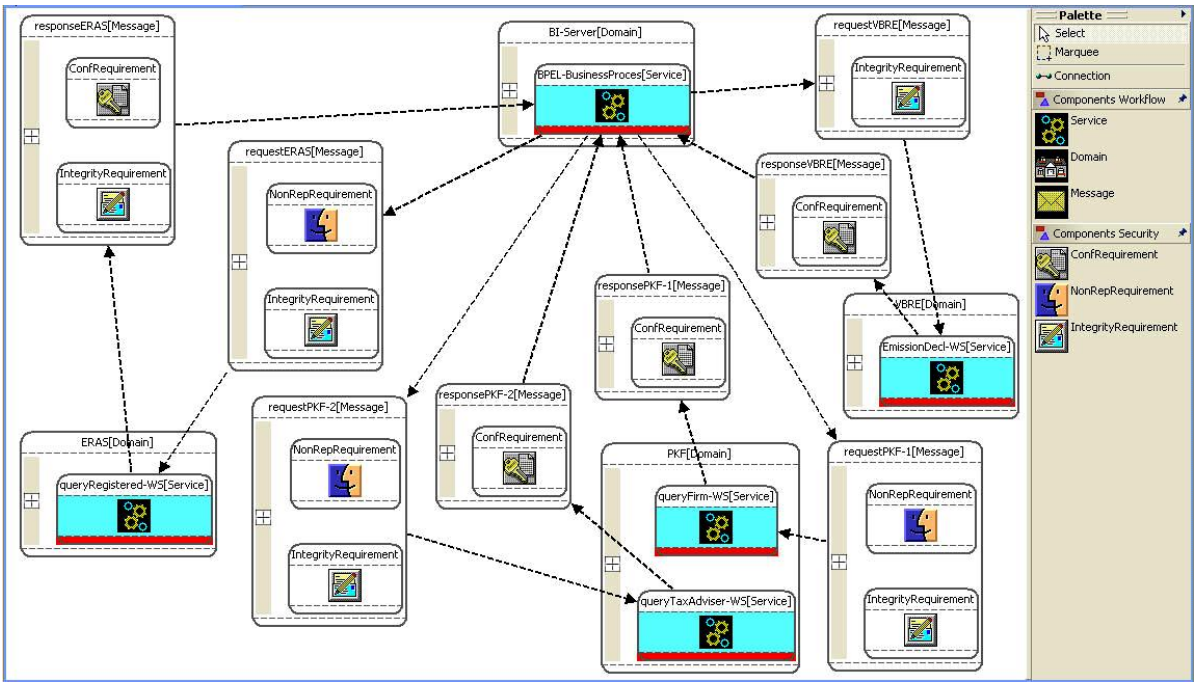

Figure 4: The Eclipse Based Modelling Tool for Web-Services Security Requirements

To model the communication between two domains one first needs to drag two domain model elements from the palette on the right onto the canvas. The bottom panel of the new domain elements is initially colored red. This indicates that some properties have to be set to make the model element valid. The properties can be set in the properties panel at the bottom of the modeling tool. For each domain a name has to be set that denotes its role in the communication. Next one needs to add services to each domain that communicate with each other by dragging services from the palette on each domain. The services need to have a name and an operation name set to be valid. These actually have to correspond to the real names of services. Now the communication needs a message sent between the two nodes. This is done by dragging a message model element on the canvas. The message has to be associated with initiating and receiving services by using the connection tool. Finally the security requirements of communication is configured by dragging security requirements from the palette onto the message. For confidentiality and integrity requirements one has to specify the XPath to the elements on which the requirements shall be enforced and also up to five namespaces. For the non- repudiation requirement no further configuration is needed. The generator can be run by right-clicking on the canvas and selecting ArcSecu Menu->DO IT. For each involved domain the generator creates one folder containing the XACML policies that need to be deployed to the policy folder of each domain.

7. Conclusions

We presented the Generic Modeling Environment (GME), a configurable modeling and program synthesis tool set, which makes the rapid and cost-effective creation of highly customized, domain-specific system design and analysis environments possible. It is highly applicable for modelling XAMCL policy generator related to security artefact for Web-services. We focus on visually modelling an XACML system to fulfil the XACML profile and automatic generation of the security infrastructure in XACML-formatted documents.

This paper has described the powerful features available for building graphical modeling tools with GEMS. GEMS focuses on allowing developers to create Eclipse-based modeling tools without writing plug-in descriptors, GMF mapping files, GEF code, or EMF code. Furthermore, GEMS provides facilities to support external specification of visualization details so that they can be managed by graphic designers and other individuals

not experienced with complex GUI coding. This Meta modelling approach reduces the cost of developing a graphical modeling tool by allowing developers to focus on the key aspects of their tool. The infrastructure to create, edit, and constrain instances of the language is generated automatically and the implementing process makes possible the separation of language development, coding, as well as "look and feel" development.

We introduce the concept of Model Driven Security for a software development process, which allows for the integration of security requirements through system models and supports the generation of security infrastructures and focuses on business logic as well as on inter-organizational workflow management.

8. Future Work

Several improvements and extensions need to be addressed in future work.

Currently our approach focuses on static design models, which are relatively close to the implementation. It is worth considering whether the efficiency of the development process of secure applications can be improved by annotating models at a higher level of abstraction (e.g. analysis) or by annotating dynamic models, e.g. state machines. Moreover, some critical questions concerning the development process are still open, e.g. how are roles and permissions identified? Beyond that, the current prototype does not yet demonstrate the platform independence of our concepts. Future work will focus on modeling security requirements and design information using dynamic models. The development process for secure systems starting with the initial analysis up to the complete secure system design will be investigated. In this context, we will examine the possibility of propagating security requirements between analysis and design models and ways to verify the compatibility of requirements and design information given at different levels.

In future work, we are developing advanced declarative programming, knowledge base, and querying functionality for GEMS, as well as addressing the challenges of maintaining model assets when the underlying meta-model changes.

References

[1] OASIS Standard. XACML profile for Web-services, http://docs.oasis-open.org/committees/documents.php?wg_abbrev=xacml

[2] OASIS Standard. eXtensible Access Control MarkupLanguage (XACML) version 2.0, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, February 2005.

[3] SAML 2.0 profile of XACML. Draft 02, 11 November 2004. - http://docs.oasisopen.org/xacml/access_control-xacml-2.0-saml_profile-spec-cd-02.pdf

[4] Ferraiolo, D. and R. Huhn. Role-Based Access Control. In Proceedings of 15th National Computer Security Conference

[5] Xin Jin, Applying Model Driven Architecture approach to Model Role Based Access Control System, Master Thesis

[6] Object Management Group. MDA Guide Version 1.0.1. [cited 2005 Nov.]; Available from: http://www.omg.org/mda/specs.htm.

[7] Policy Generator, Taufiq Rochaeli, TUD SEC, Ruben Wolf, Fraunhofer-SIT, February 10, 2006.

[8] Simplifying the Development of Product-line Customization Tools via Model Driven Development, Jules White and Douglas Schmidt, Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, USA

[9] GEMS EMF Intelligence Tutorial, http://wiki.eclipse.org/GEMS_EMF_Intelligence_Tutorial

[10] GEMS EMF Intelligence Tutorial with Mixed Constraints, http://wiki.eclipse.org/GEMS_EMF_Intelligence_Tutorial_with_Mixed_Constraints

[11] GEMS Metamodeling Tutorial, http://wiki.eclipse.org/GEMS_Metamodeling_Tutorial

[12] The Generic Eclipse Modeling System (GEMS), By Jules White

[13] openArchitectureWare 4.2 Fact Sheet, Markus Völter, voelter@acm.org Date: September 3, 2007

[14] GrTP: Transformation Based Graphical Tool Building Platform, Institute of Mathematics and Computer Science, University of Latvia

[15] Building Tools by Model Transformations in Eclipse, University of Latvia, Audris Kalnins, Oskars Vilitis1, Edgars Celms