A Web-Based Collaborative Metamodeling Environment with Secure Remote Model Access*

Matthias Farwick[1], Berthold Agreiter[1], Jules White[2], Simon Forster[1], Norbert Lanzanasto[1], and Ruth Breu[1]

[1] Institute of Computer Science University of Innsbruck, Austria
{matthias.farwick,berthold.agreiter,simon.forster,
norbert.lanzanasto,ruth.breu}@uibk.ac.at
[2] Electrical Engineering and Computer Science Vanderbilt University, Nashville, TN, USA
jules@dre.vanderbilt.edu

Abstract. This contribution presents GEMSjax – a web-based metamodeling tool for the collaborative development of domain specific languages. By making use of modern Web 2.0 technologies like Ajax and REST services, the tool allows for simultaneous web browser-based creation/editing of metamodels and model instances, as well as secure remote model access via REST, which enables remote model modification over a simple HTTP-based interface. This paper describes the complex technical challenges we faced and solutions we produced to provide browser-based synchronous model editing. It further explains on the XACML-based access control mechanisms to provide secure remote access to models and model elements. Additionally, we highlight the usefulness of our approach by describing its application in a realistic usage scenario.

1 Introduction

Nowadays, there exist a multitude of different modeling tools for a variety of purposes. The wide adoption of model-driven techniques further stimulates the creation of domain specific modeling languages. Furthermore, in today's globalized economy engineering teams are often geographically dispersed to cut down costs, bring together expertise, or to explore new markets. However, to allow such teams to collaborate in an efficient manner, specific tools are needed which support the collaborative way of working. Modeling is a well-established task in software engineering and enterprise architecture today. In this context, modeling tools are used, for example, to communicate software architecture or to model the IT-landscape of large organizations. Contemporary research studies have shown evidence that complex projects conducted by such virtual teams are less

* This work was partially supported by the Austrian Federal Ministry of Economy as part of the Laura-Bassi – Living Models for Open Systems – project FFG 822740/QE LaB.

B. Benatallah et al. (Eds.): ICWE 2010, LNCS 6189, pp. 278–291, 2010.

successful than geographically concentrated teams [5]. The difficulty of sharing knowledge in dispersed teamwork has been identified as one of the main reasons for such failures [4,3]. Therefore, collaborative tools have to be created that aid the experts in sharing knowledge, and working together on solutions. Another fact which is mostly not respected by current modeling tools are concepts that allow feeding back information from the real world into models, thus keeping the model (partly) in–sync with the real world. To cope with the two requirements of collaborative modeling and remote model access, we present GEMSjax[1], a web-based metamodeling tool leveraging the full technological capabilities of Web 2.0. GEMSjax has been built from the ground up having collaborative modeling and bidirectional information flow in mind, e.g. by supporting concurrent browser-based modeling and collaborative chat.

The tool allows for developing metamodels from which web-based graphical editors for the corresponding domain specific language (DSL) are generated. These generated editors run in the same browser-based graphical environment, hence there is no need for users to install software for participating in the modeling process. Many of the features, such as look-and-feel customization via stylesheets, dynamic styles reflecting attribute changes, as well as the remote model access, are inspired by the Generic Eclipse Modeling System (GEMS [13]). Apart from the metamodeling capabilities, the tool employs a novel approach to remote model access by providing a REST-based (REpresentational State Transfer [6]) remote interface for model elements. The interface can be used to easily integrate GEMSjax models into other applications and to provide live updates of model elements by remote applications. These updates can be used to keep the model partly in–sync with what it represents in the real world. For example, in a model of a server landscape, the remote interface can be used to signal when a new server is added to the system or when a server is down. A malfunctioning server could be, for example, drawn in red, or, in case of the addition of a new server, a new model element can be added to the model remotely.

We faced the following implementational and conceptual challenges because of the web-based nature of our solution: i) browser-based manipulation of models, ii) synchronization of simultaneous model changes by different clients, iii) access control management of the remote interfaces. To provide the browser-based manipulation of models we created a client-side representation for them. In order to tackle the synchronization problem, we made use of a bidirectional push protocol. In order to cater for adequate security protection of the interfaces to models, we developed a sophisticated access control architecture based on the eXtensible Access Control Markup Language (XACML [9]). This contribution focuses on our solutions to the aforementioned challenges.

The remainder of this paper is structured as follows. The next section motivates the need for GEMSjax by giving a short introduction on metamodeling and summarizing the requirements. Section 3 outlines a typical usage scenario that requires the features of GEMSjax. In Section 4 we provide a high-level

[1] GEMSjax is a short form of GEMS(A)jax, where GEMS is the Eclipse-based tool whose features GEMSjax brings to the web via Ajax (see Section 4).

overview of the challenges we faced and solutions we created to provide browser-based graphical model editing. After that, Section 5 focuses on the secure REST model interface and the XACML-based architecture to enforce access control on models and model elements. The final section concludes the paper by referring to related tools and pointing out directions of future work.

2 Motivation

To justify the need for the tool presented here, we first cover background in metamodeling as this is the main foundation this work builds on. After that, we identify the requirements for such at tool, to prepare the reader for the expected solution realized by GEMSjax.

2.1 Metamodeling

Today, software engineers have several possibilities in order to create models for a given domain. The predominant choice is still the use of General Purpose Modeling Languages (GPML) like the Unified Modeling Language (UML)[2]. The metamodel of such languages predefines their syntax and partially their semantics. In order to customize, e.g. UML, one needs to utilize the stereotyping mechanism, so that customized models remain compatible to their base language. With this mechanism one can apply new semantics to meta-elements. However, the underlying syntax remains static and needs to be manually adjusted via the use of the Object Constraint Language (OCL). Restricting the metamodel in such a way is very cumbersome and error-prone, leading to inconsistent models. Another alternative is creation of a Domain Specific Modeling Language (DSML). Opposed to a GPML, a DSML is specifically created to model selected domain aspects. Here, a metamodel is created from a generic meta-metamodel that specifies the syntax and semantics of the desired language, thereby only describing what is allowed in the language, and implicitly prohibiting everything that is not specified. With UML, on the other hand, everything that is not explicitly prohibited is allowed to be modeled. The comparison of the two above-mentioned modeling processes is shown in Figure 1.

Therefore, UML is more suitable for application areas that are closely related to the core competence of UML, like Software Engineering, where the syntax and semantics only need to be slightly adapted. DSMLs however, allow for a precise language definition *without* the need to restrict a large metamodel. Furthermore, by using well-designed DSMLs, the modeling process can be considerably speeded up compared to GPMLs like UML [8,7].

2.2 Requirements Analysis

Before we start with the description of the tool and the problems we faced, we first identify its requirements. As many design decisions are motivated by the

[2] `http://www.uml.org/`

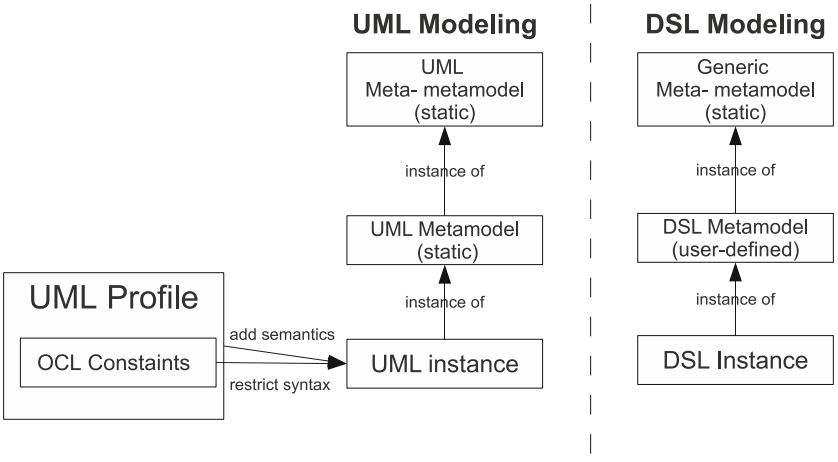

Fig. 1. UML Profiling vs. Domain Specific Modeling

requirements, this section clarifies why specific approaches have been selected. For a quick overview, Table 1 shows all requirements in a condensed representation.

Today's economy becomes more and more globalized, e.g. companies have subsidiaries distributed all over the world. This leads to the need for synchronization and effective collaboration over distance. We want to provide a tool supporting this collaboration. Collaboration should be as easy as possible for the partners, i.e. instant availability without the need for installing software packets (*R1*). Furthermore, it is likely that due to the geographical dispersion, heterogeneous environments will be encountered. Because of this, our tool should be platform-independent wherever applicable (*R2*). With GEMSjax we want to provide a tool that helps domain experts to express their knowledge as directly as possible. Domain-specificity is aimed at targeting exactly the questions of the domain, without blurring the important information by side information. For this reason, a further requirement to GEMSjax is the ability to create domain specific languages via metamodeling (*R3*). As already mentioned, an additional requirement to our tool is the possibility to support a collaborative way of modeling. This collaboration should allow several domain experts to view and edit models simultaneously (*R4*).

A known problem of models is that they are often created in the design phase only, and not maintained anymore at later stages. However, if the information captured in the models changes later on, the model and the real world are out-of-sync. This is certainly not desired, and should be avoided to derive greater benefit from models, cf. [2]. Such updates are not necessarily executed by humans, e.g. an information system which has just been booted up can update its status in the model autonomously. For this reason, GEMSjax aims to provide a way for updating models remotely, even without using a graphical modeling tool (*R5*). Naturally, this interface needs to be protected against unauthorized usage.

Table 1. GEMSjax requirements

R1	Instant availability for users, without installing large packages.
R2	Platform independence.
R3	Ability to create domain specific languages.
R4	Simultaneous model access.
R5	Secure remote model API for querying/manipulating models.

3 Usage Scenario

The IT-landscape of the large global enterprise Example Corp. is distributed over several continents, technologically heterogeneous, and poorly documented, since it evolved over several decades and was changed by many different individuals. Additionally, it is often not clear to the management of Example Corp. how each (technical) component (e.g., servers, information systems, etc.) in the IT-landscape contributes to the core business of the enterprise. Therefore, the CIO decides to start an Enterprise Architecture Management (EAM) effort, in order to document and model the global IT-landscape, standardise used technology (e.g. only Apache Tomcat version 6 should be in use), plan infrastructure change, and to analyze how each component contributes to the business goals of the company. It is also decided that in the long run, the IT-landscape model should be coupled with the actual run-time infrastructure in order to always have an up-to-date view of the infrastructure.

The first step in the EAM initiative of Example Corp. is the *metamodel definition* (R3) of the enterprise's IT-landscape and the business functionalities (e.g. selling hotel bookings, selling cars). *Due to the distributed nature of Example Corp. not all stakeholders of the company's IT can gather for a physical meeting to discuss the metamodel. Therefore, a web-based meta-modeling solution like GEMSjax is chosen (R4).* The IT-responsibles at each data center meet in the virtual modeling environment, where they can collaboratively create the enterprise metamodel, and communicate via a chat to discuss ideas. Furthermore, the dynamic graphical appearance of each model element is defined. For example, a server, whose workload reaches a certain threshold dynamically appears in red.

After several weeks of discussion, an agreement on the metamodel is found that is capable of expressing all necessary IT and business assets of the company. An instance of this metamodel is created in GEMSjax and the stakeholders at each site insert their IT-infrastructure in the global model. Where interfaces between two data centers exist, these model elements are also collaboratively created in GEMSjax.

Finally, a satisfying representation of the enterprise IT-landscape is created. However, this representation is hard to keep up to date without unjustifiable manual labor. *In order to reduce this workload, some of the infrastructure devices are equipped with agents that communicate their existence and state to the GEMSjax model via its REST interface (R5).* The high security standards of Example Corp. are met by the XACML-based access control engine of GEMSjax.

4 The GEMSjax Metamodeling Tool

To enable distributed, collaborative (meta-)modeling, needed for the usage scenario described before, we developed GEMSjax, a browser-based graphical model editor. The web-based nature of the tool allows for instant availability and platform independence, as everything needed on the client-side is a web browser with Javascript capabilities (R1 & R2). GEMSjax provides a graphical modeling view for Eclipse Modeling Framework (EMF)[3] models. EMF is an Eclipse project that provides the means to work on top of a structured data model, include code generation and manipulation interfaces. GEMSjax is written in Java and the client-side browser editor is compiled from Java into Javascript and HTML using the Google Web Toolkit (GWT)[4]. The server-side of GEMSjax is built on Java servlets. Figure 2 shows its web interface with the classical modeling tool setup. The left side shows the metamodels and model instances specific to the logged-in user. The modeling canvas is located in the center and has nested tabs for metamodels and different views for them. Model elements can be dragged from the palette on the right onto the canvas. Each model instance has its own chat where users, which have the right to view or edit the model, can post messages. Attributes of a selected model element can be viewed and edited in the center-bottom panel. Table 2 summarizes the key features of GEMSjax to give the reader a quick overview.

The remainder of this section provides a high-level overview of the challenges we faced and solutions we created to provide browser-based graphical model editing.

4.1 Client-Side Manipulation of EMF Models

One of the key challenges of developing GEMSjax was establishing a method for building a client-side in-memory representation of a server-side EMF model. EMF models, themselves, cannot be transported to the client via HTTP and loaded into memory as Javascript (a future alternative might be the recent Eclipse project proposal JS4EMF[5]). To address this challenge, we created a generic Javascript object graph for representing EMF models in memory on the browser. The client-side memory representation manages structural constraints in the model, such as allowed parent/child relationships or valid associations between model elements. The reason for checking such constraints on the client-side instead of on the server is to avoid one roundtrip. This enhances the user-experience because immediate feedback is provided without the need to wait for a server response on each action. If, in the future, complex constraints are introduced, e.g via OCL, these should be evaluated on the server-side to ensure model consistency. Our solution allows for leveraging server-side computation power and reusing existing constraint libraries which are currently non-existent for client-side Javascript.

[3] `http://www.eclipse.org/modeling/emf/`
[4] `http://code.google.com/webtoolkit/`
[5] `http://www.eclipse.org/proposals/js4emf/`

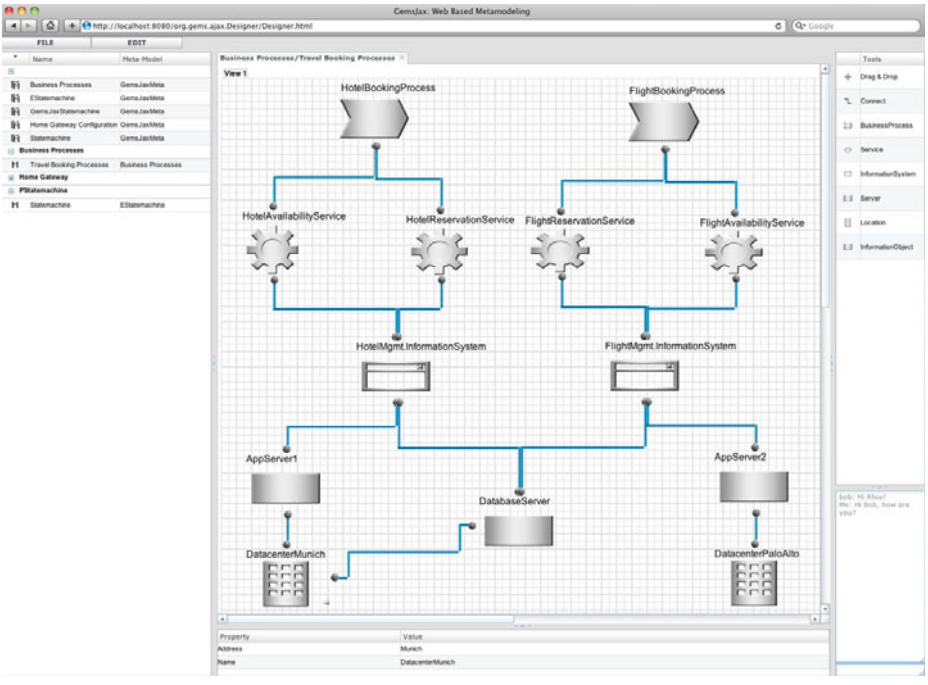

Fig. 2. Screenshot of the GEMSjax web-interface

When a model is first opened, a Java servlet on the server reads in the EMF model from disk and translates the model into an equivalent JavaScript Object Notation (JSON) representation that is sent to the client to be loaded into memory using our generic Javascript object graph framework. The server-side also associates a unique ID with each EMF and client-side Javascript object in order to provide a mapping from EMF objects to Javascript objects and vice-versa. The structural rules from the Ecore metamodel of the EMF model are also extracted into a JSON representation and transported to the client-side to enforce basic structural constraints, such as allowed containment relationships.

4.2 Bidirectional Client/Server Model Synchronization

For implementing the simultaneous model access (R4), a bidirectional client/server synchronization scheme is used. HTTP is designed for a request/response style communication between a client and remote server. A key challenge we faced was determining how to *push* changes triggered in the server-side EMF model to the client-side Javascript representation. In situations where a model edit originated from the client-side, any updates, such as triggered model transformations, performed on the server can be returned in the HTTP response

Table 2. GEMSjax key features

Key Feature	Description
Metamodeling	Typical metamodeling possibilities: (abstract-) classes, attributes, connections, inheritance
Model Instances	Instantiate metamodel in same web interface
Customization	Instance customization via stylesheets
Dynamic styles	Dynamic styles of model elements depending on attribute values
Remote Interface	Model modification via REST interface
Access Control	Role Based Access Control of graphical and REST interface
Collaborative Modeling	Simultaneous modeling
Chat	One chat instance per model instance
Export	Exports EMF models, e.g. for model transformation

to the client. However, changes in the model that originate in the server *outside* of an HTTP response to a client request, such as time-based triggers, incoming REST API calls, or edits from modeling collaborators, cannot be pushed to the browser using standard HTTP approaches.

To address this limitation of standard HTTP request/response architectures, GEMSjax uses the bidirectional HTTP push protocol Cometd[6] to allow updates from the server to be delivered to the client browser. GEMSjax uses an event bus built on top of the Cometd protocol to communicate changes between clients and the server. Synchronization between clients editing the same model is maintained by broadcasting model edits on the event bus to each connected client. Each client compares the state change in incoming events to the model to determine if the event represents a duplicate change. A priority scheme is used to reconcile and rollback conflicting changes. Also, information about which elements are currently selected by other modelers can be pushed to the other clients to avoid conflicting changes a priori. However, the effectiveness of our conflict avoiding/resolution approach needs further investigation.

The sequence diagram in Figure 3 shows an example of the bidirectional communication with two clients connected to the server. When a client connects to the server, it first gets the most current version of the model by calling the `getModelPackage(modelID: String)`-method. Note that all calls originating from clients are asynchronous calls, s.t. the behavior is non-blocking. For every client to receive updates that originate from a different source than itself, it makes a call to `receiveModelChange()`. This call opens a connection to the server and leaves it open until either an update has to be delivered to the client, or a timeout occurs. In our example, Client1 updates the model (step 7) and afterwards Client2 receives the updated version (step 9) as response of the call in step 6. After client2 receives the update it re-opens the connection to the server in the final step 10. Note that this technique avoids frequent polling. Furthermore, updates are delivered to all clients immediately after the server changes the status of a model.

[6] `http://cometdproject.dojotoolkit.org/`

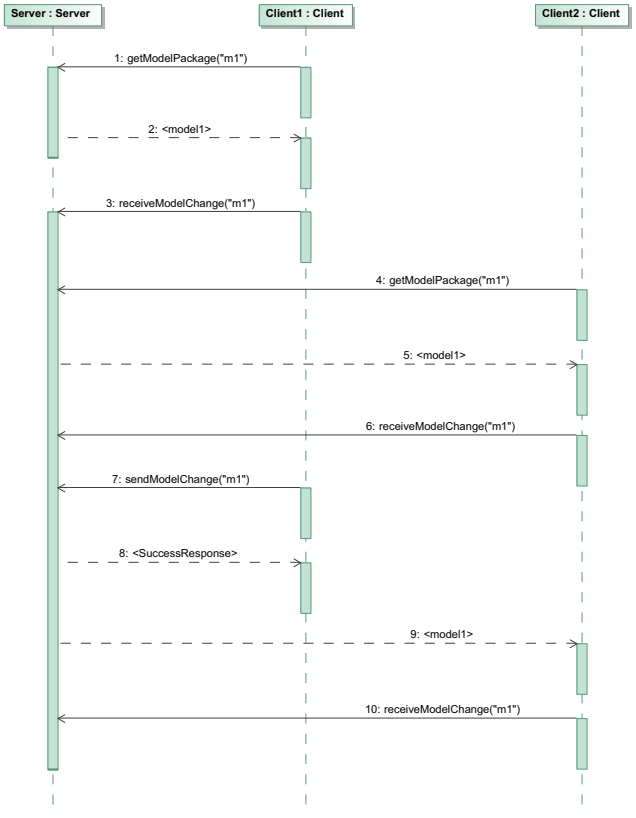

Fig. 3. Client-push protocol example with two clients connected to the server

4.3 (Meta-)modeling Lifecycle

The metamodel representations in GEMSjax are managed on the server-side. Figure 4 shows the lifecycle of the involved metamodels and their corresponding instantiations. On top of the figure is the GEMSjax metamodel which is predefined as an Ecore metamodel (1).

In the metamodeling step an EMF instance (2) of the GEMSjax Ecore metametamodel is created according to the requirements of the new DSML. This realizes our requirement R3. Each time the metamodel is saved on the client side (web–browser) the model is serialized to its EMF representation on the server. Once the metamodel is finished, it is transformed again to act as an Ecore metamodel (3) for new DSML model instances. This is achieved via JET templates[7] that transform the EMF representation of the metamodel into an Ecore metamodel. Finally, instances of that DSL-specific Ecore metamodel can be created with the client (4).

[7] `http://wiki.eclipse.org/M2T-JET`: JET is a template-based code generation framework, which is part of the Eclipe Model to Text (M2T) project.

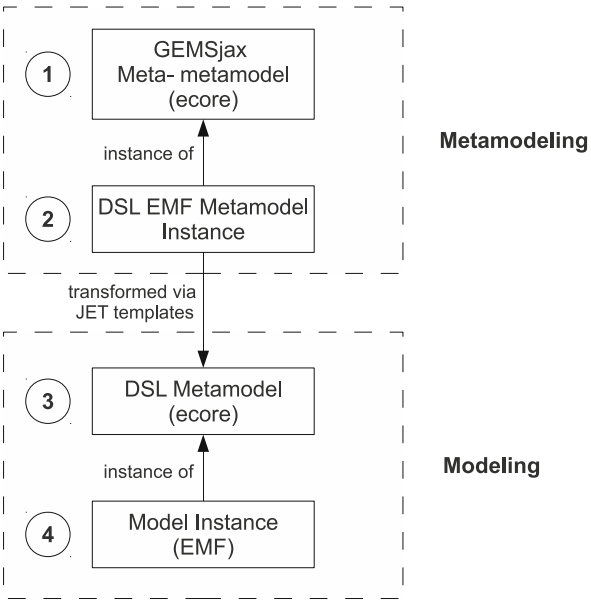

Fig. 4. Image showing the modeling lifecycle from the GEMSjax metamodel to a domain specific model instance

5 Secure Remote Model Access

As mentioned earlier, one of the key-features of GEMSjax is the easy-to-use REST interface of the models. Using this interface allows developers to integrate information of models in their own tools or update the models, because the HTTP protocol is available in practically any programming environment. This way, models can become actual configuration artifacts and represent the real state of a system.

5.1 The REST Interface

The REST API treats models as hierarchical resources that can be accessed and manipulated via the basic HTTP operations GET, PUT, POST and DELETE. It uses the hierarchical property of URIs to describe containment of model elements. The *GET* operation is used to retrieve information about model elements, *PUT* to create new elements, *POST* and *DELETE* to change and delete elements respectively. Table 3 exemplifies the usage of the simple interface calls.

By specifying return mime-types the user can choose the return type of the operation, e.g. XML or JSON. This allows for a flexible client implementation. Input values for the PUT and POST commands are transmitted via simple key-value pairs in the body of the messages. Responses conform to the HTTP specification, e.g. by returning *201 (Created)* for a PUT request containing the URI of a newly created model resource.

Table 3. Examples of REST HTTP Commands

Operation	HTTP Command and URL
Get list of available models	GET http://.../gemsjax/models
Get attributes of an element	GET http://.../gemsjax/modelID/.../elementID/attributes
Get all meta element names of model	GET http://.../gemsjax/modelID/meta
Create new element	PUT http://.../gemsjax/modelID/.../metaType
Update Element	POST http://.../gemsjax/modelID/.../elementID
Delete Element	DELETE http://.../gemsjax/modelID/.../elementID

5.2 Access Control Architecture

To constrain access to model elements we incorporate an *Access Control Layer* in our architecture (cf. Fig. 5). With each request a user needs to provide credentials in the form of a username/password combination over HTTPS. As mentioned before, there are two possibilities to access GEMSjax models – the *REST interface* and the graphical interface of the modeling tool (*GWT RPC interface*). Calls via the REST API are first translated to the actual action to be taken on the model by the *URI Mapper*. After that, requests are forwarded to the *Request Handler* which provides a model interface independent of the access method. These actions could immediately be executed on the model, however the request first has to pass the Access Control Layer.

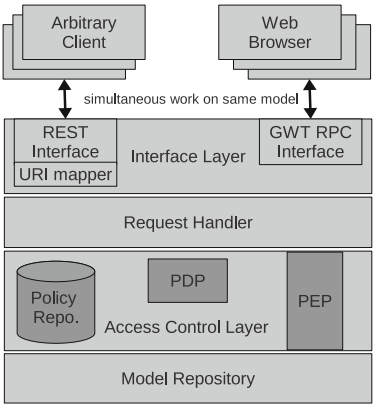

Fig. 5. Access Control Architecture

The access control layer is based on the XACML Target Architecture (cf. [9]), since it is a proven access control architecture with several open-source implementations. In an XACML architecture, requests are generally first intercepted by the *Policy Enforcement Point* (PEP). It forwards the request context (e.g. user credentials, requested action) to the *Policy Decision Point* (PDP). The PDP decides on the access rights according to policies stored in a policy repository. This decision is then enforced by the PEP. A typical example for an XACML

```
<Policy PolicyId="PolicyId123" RuleCombiningAlgId="permit-overrides">
 <Target><Subjects><AnySubject/></Subjects>
   <Resources><Resource>
     <ResourceMatch MatchId="isSubURI-or-equal">
      <AttributeValue DataType="anyURI">
       http://.../gemsjax/exampleModelID</AttributeValue>
    <ResourceAttributeDesignator DataType="anyURI" AttributeId="resource-uri"/>
   </ResourceMatch></Resource></Resources>
  <Actions>
    <ActionMatch MatchId="string-equal">
     <AttributeValue DataType="string">update</AttributeValue>
     <ActionAttributeDesignator DataType="string" AttributeId="action" />
    </ActionMatch>
  </Actions></Target>
 <Rule Effect="Permit" RuleId="updateRule">
  <Target />
  <Condition FunctionId="string-equal">
   <Apply FunctionId="string-one-and-only">
    <SubjectAttributeDesignator DataType="string" AttributeId="role" />
   </Apply>
   <AttributeValue DataType="string">admin</AttributeValue>
  </Condition>
 </Rule>
</Policy>
```

Fig. 6. An example XACML policy

policy in GEMSjax is depicted in Figure 6. First, a rule combining algorithm is set in the initial $< Policy >$ element. This algorithm governs the final result of the policy evaluation depending on the evaluation of $< Rule >$ elements. In this case a $PERMIT$ evaluation of a rule overrides the $DENY$ result of a previous rule. The following $< Target >$ element specifies to which subject, resource, and requested action the policy applies. Here, it applies to actions of the type `update`, on the model resource with ID `exampleModelID`. Finally, the *Rule* element states that for the user with the role `admin` access should be permitted.

Depending on the PDP's decision, the Policy Enforcement Point will forward the action to the *Model Repository* or block access. The creator of a metamodel is in charge of granting permissions to registered users of the system. This entails that GEMSjax enforces a combination of Role-based Access Control (RBAC) and Discretionary Access Control (DAC)[8].

Note that the XACML policies can be very fine grained down to the attribute level, and are very flexible, e.g. to allow another group of users to update the model it suffices to add another rule element to the policy. Also policies can be used for complex requirements like delegation of rights, if, e.g., a user wants to invite an expert that only has read access to the model for consulting purposes. The REST interface to GEMSjax and the access control architecture described here, form the realization of requirement R5.

6 Related Work and Conclusion

In this work we presented GEMSjax a web-based metamodeling tool for the collaborative development of domain specific languages. We highlighted its

[8] In Discretionary Access Control the owner of an object has the right to delegate permissions on an object to other users.

usefulness in a usage scenario and described the technological challenges and our solutions for implementing a metamodeling tool in a web enabled manner. We also presented a novel approach to secure (remote) model access, based on REST and XACML.

Related literature describes several collaborative modeling approaches, none of which combines all three key features of GEMSjax: the secure REST model interface, web-based DSL definition, and simultaneous collaborative modeling.

In his work on COMA [10], Rittgen describes a collaborative modeling tool for UML models, that provides a voting mechanism to achieve consensus on a model. Opposed to our work this tool does not provide the means to create domain specific languages and has no means to enforce access control on models. The commercial modeling tool MagicDraw Teamwork Server [1] allows for collaborative modeling by providing a locking mechanism for models but does not provide a REST interface and strictly operates on UML models. Many of the features of GEMSjax are inspired by GEMS [13], however GEMS is not browser-based, does not support collaborative modeling, and does not provide security for its RPC interface. The Eclipse-based Graphical Modeling Framework (GMF)[9] also allows for graphical metamodeling, but is relatively more complicated to configure and does not provide web-based and collaborative modeling. The research prototype SLIM, presented in [12], uses similar technology to GEMSjax and also aims for a lightweight collaborative modeling environment. However, it is only able to represent UML class diagrams, which contradicts to our requirement to create domain specific languages. Moreover, SLIM does not offer a remote model API which eliminates the usage of the tool where updates should not be made by human users only.

Our future work in this area will include further development of the tool as well as experiments. Specifically, we will work on an Eclipse integration, that will enable to use all Eclipse features while editing GEMSjax models in the Eclipse browser tab. Collaborative modeling environments expose a number of further issues to solve, like locking or versioning. Future work will also comprise the integration of existing solutions for such problems (e.g. [11]). We will investigate on the integration of model voting, concurrency issues and code generation techniques. We also plan a collaborative modeling experiment with dispersed teams of students in the US and Austria to get further insight on the effectiveness of our collaborative modeling approach.

References

1. Blu Age: MagicDraw TeamWork Server (2009), `http://www.bluage.com/?cID=magicdraw_teamwork_server`
2. Breu, R.: Ten principles for living models - a manifesto of change-driven software engineering. In: 4th International Conference on Complex, Intelligent and Software Intensive Systems, CISIS-2010 (2010)
3. Conchúir, E.O., Ågerfalk, P.J., Olsson, H.H., Fitzgerald, B.: Global software development: where are the benefits? Commun. ACM 52(8), 127–131 (2009)

[9] `http://www.eclipse.org/modeling/gmf`

4. Cramton, C.D.: The mutual knowledge problem and its consequences for dispersed collaboration. Organization Science 12(3), 346–371 (2001)
5. Cramton, C.D., Webber, S.S.: Relationships among geographic dispersion, team processes, and effectiveness in software development work teams. Journal of Business Research 58(6), 758–765 (2005), http://www.sciencedirect.com/science/article/B6V7S-4BM92C5-4/2/60cbbf7fa88eb389c6e745f355acca58
6. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine, Irvine, California (2000)
7. Frank, U., Heise, D., Kattenstroth, H., Fergusona, D., Hadarb, E., Waschkec, M.: ITML: A Domain-Specific Modeling Language for Supporting Business Driven IT Management. In: DSM '09 (2009)
8. Luoma, J., Kelly, S., Tolvanen, J.: Defining Domain-Specific Modeling Languages: Collected Experiences. In: Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modeling, DSM '04 (2004)
9. OASIS: eXtensible Access Control Markup Language (XACML) Version 2.03. OASIS Standard (February 2005)
10. Rittgen, P.: Coma: A tool for collaborative modeling. In: CAiSE Forum, pp. 61–64 (2008)
11. Schneider, C., Zündorf, A., Niere, J.: CoObRA-a small step for development tools to collaborative environments. In: Proc. of the Workshop on Directions in Software Engineering Environments (WoDiSEE), Edinburgh, Scotland, UK (2004)
12. Thum, C., Schwind, M., Schader, M.: SLIM – A Lightweight Environment for Synchronous Collaborative Modeling. In: Schürr, A., Selic, B. (eds.) MODELS 2009. LNCS, vol. 5795, pp. 137–150. Springer, Heidelberg (2009)
13. White, J., Schmidt, D.C., Mulligan, S.: The Generic Eclipse Modeling System. In: Model-Driven Development Tool Implementer's Forum at the 45th International Conference on Objects, Models, Components and Patterns (June 2007)