



Metada Metarepository Architecture

Overview

Author: Metada Team

Contents

1 Metada Metarepository Architecture Overview.....	3
2 Metametamodel (m2_mta).....	3
2.1 Model Editors.....	3
2.2 Model Storage.....	4
2.3 Model Versioning.....	4
2.4 Model Validation.....	4
2.5 Designing New Metamodels.....	5
3 Repository Metamodel (m2_rep_mta).....	5
3.1 Authentication.....	6
3.2 Authorization.....	6
4 Genet Metamodel (m2_genet_mta).....	6

1 Metada Metarepository Architecture Overview

Technically, Metada Metarepository is a platform for execution of metamodels. It is an architecture framework that enables a metamodel to be inserted and executed. Each metamodel contains description of model entities it allows in its conformant models and (RESTful) services that it provides.

Apart from domain-specific off-the-shelf metamodels, Metada Metarepository contains also few core metamodels that make-up the heart of the metamodeling solution. *Core metamodels* are the Metametamodel (m2_mta), Repository Metamodel (m2_rep_mta), and Genet Metamodel (m2_genet_mta). The individual core metamodels are discussed below in their respective sections.

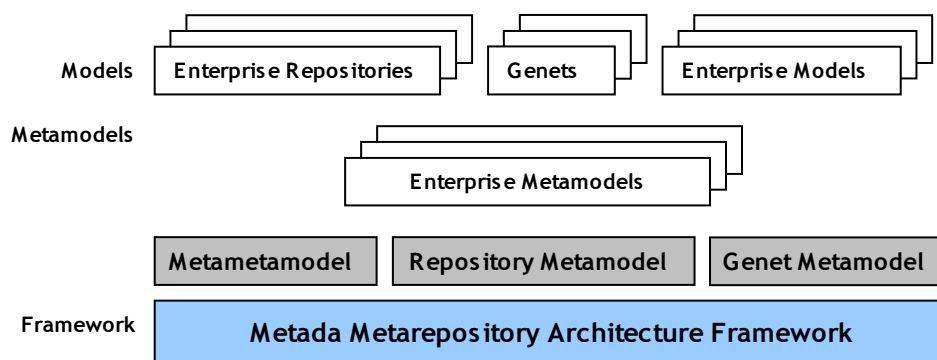


Figure: Metada Metarepository Architecture Overview

Metada Metarepository architecture framework is a web-based environment written in 100% pure Java that may run on any web server that supports the Java Servlet API (i.e. Apache Tomcat, BEA WebLogic, etc.). The RESTful services defined in metamodels interpret instances of models. Interpreters are packaged directly with metamodels and are in the form of either Java code or server-side scripted code (i.e. Ruby, JavaScript, etc.).

2 Metametamodel (m2_mta)

The m2_mta metametamodel is an EMOF compliant metamodel that defines how metamodels may be composed and what services are available for metamodels (model storage, editing, validation, versioning, etc.). Models are stored in an SQL database (MySQL, MSAccess, Oracle, etc.) and versioned in Subversion. Metametamodel contains an interpreter that interprets metamodels and automatically creates databases for models and generates all SQL queries on-the-fly. The following subsections discuss the available interpreters.

2.1 Model Editors

Metada Metarepository is accessed via regular web browser without any special plug-ins or additions. Model editors take care of converting any XML results of the services into HTML format in which replies are sent back to the user's browser. Conversion from XML data to HTML is done via XSLT templates. Each model editor knows what templates it needs to use to properly convert its results to HTML.

Model editors include multi-lingual support based on the user's browser language preferences. The preferred language is checked and if the XSLT template in that language exists it is used to present the data produced by the editor. If the user's language is not supported then default language is used.

Model editors are *interpreted* on-the-fly from corresponding metamodels. Interpreted artifacts include the screen flows, lists, forms, flaps, and navigations in XML and XSLT formats. Model editors may be manually appended with program code to enable special-purpose functionalities. Metamodels also define menu structures for editors of their models.

2.2 Model Storage

The m2_mta metametamodel contains XML Data Access (XDA) interpreter that enables models of various metamodels to be stored in a relational database. Queries are placed and results are obtained in the form of XML documents. The RESTful model storage services offer three operations on model entities:

- **GetData (GET)** – get query that gets data from a database into an XML document (database SELECT statement)
- **UpdateData (PUT)** – update or insert request either inserts or updates data in a database (INSERT or UPDATE database statement)
- **DeleteData (DELETE)** – delete request deletes data in a database (DELETE database statement)

XDA Interpreter Philosophy

To understand how XDA works and how queries should be built a look needs to be taken at how XDA does its job. Here are some of the major points:

- XDA dynamically generates SQL queries that it sends to a database.
- One XDA query may result in several SQL queries, and thus one XDA query may take data from or update data in several database tables at a time.
- Queries are prepared from cached and pre-processed metadata (data about the data that the query is working with).
- The metadata needs to be specified for the query before the query may be placed (the necessary metadata are defined in a corresponding metamodel that confirms to the metametamodel)

2.3 Model Versioning

Models are primarily stored in a version control system (Subversion) and are being loaded to databases only for convenience purposes. The metametamodel includes an interpreter that offers services that take care of checkouts and commits of models, model comparison and synchronization, and model branching.

2.4 Model Validation

Model validation interpreter allows policies to be defined within metamodels to constrain models. The policies are realized as XSLT templates that traverse models and check for

specified conditions. Each policy may result in errors or warnings that are presented in the form of a report.

2.5 Designing New Metamodels

Metada Metarepository may execute packaged off-the-shelf metamodel (i.e. the Frontend Metamodel or SOA Services Metamodel). Nevertheless, it is often useful to design a specific metamodel to fit the customer's needs more exactly. The Metada Metarepository metamodel (m2_mta) defines what entities and relations metamodels may contain, and how the metamodeling GUI should look like. Therefore m2_mta fully describes the environment for designing new metamodels. From m2_mta it may be seen that for each new metamodel the designer needs to define mainly the concepts with properties and associations. Secondly, the designer specifies constraints of the modeling environment and the modeling GUI.

Metamodeling is a difficult task, because one needs to figure out the simplest set of model entities that are needed in models while allowing the models to be strong enough to express all the details that are needed. Metamodeling is very similar to creating a UML class diagram with an object model. The difference is only that the classes in metamodeling are classes that define other classes. Therefore metamodeling is moved one level of abstraction higher, but all the modeling rules, such as need for clean design, remain the same.

Steps:

- **Define concepts**
Concepts are the meta-classes for the models that the metamodel needs to enable. Concepts tell what model entities will be supported. For example in modeling business services concepts like a Service, Message, or Data Object may be defined. For modeling forms concepts like Forms with fields and controls may be defined.
- **Define metamodel validations**
Validations are rules and policies that have to be obeyed in individual instances of models that confirm to the metamodel. The validations may be executed by modelers to check if their models confirm to the rules and policies (validation reports are produced).
- **Implement generation models and interpreters**
When metamodel concepts are defined then the modeling GUI is instantly available (since the Metada Metarepository directly executes the metamodel). If the models are to be interpreted or code or reports are to be produced from models then generation models or interpreters may be defined.

The above steps are iterated as long as the metamodel is built and finalized.

3 Repository Metamodel (m2_rep_mta)

Repository models that conform to the repository metamodel define collections of models, their relations to other models, user groups and their rights to models. Repository models serve as megamodels since they define models and their relations to other models. The most important relation is the "metamodel" relation, which specifies what is the metamodel of a given model. Repository models also define specific repository GUI skins and repository menus. Repository metamodel contains authentication services for LDAP and OpenID authentication.

3.1 Authentication

Metada Metarepository has very strict user access control and separates the act of authentication from the authorizations an authenticated user may obtain to models. There are two methods of authentication available:

- **LDAP Authentication**
This type of authentication is mainly used in an enterprise setting where all users are assigned their identity and there is an identity server they use to authenticate themselves. In this case metarepository requests a user login and password and over an encrypted connection authenticates the user on the central identity server.
- **OpenID Authentication**
This modern type of authentication allows users to have only one identity to access various applications on the web. The user identities are managed by OpenID providers (i.e. www.myopenid.com) and metarepository thus does not have to store and manage any central database of users and their passwords. After entering his/her OpenID the user is redirected on pages of their provider where they approve sign-in to the application.

3.2 Authorization

Repository models define the level of access users or user groups have to individual models within the given repository. Read or read-write access level is distinguished either for the full model or for its individual parts.

4 Genet Metamodel (m2_genet_mta)

Metada Metarepository generation network (Genet) is able to obtain data from models and with a set of transformations it may produce useful outputs. For example, a model may contain management of articles and chapters in a user manual and generation node (gen) may be used to generate the manual's PDF or its static HTML representation. The same may happen with models that are generated into program code or configurations.

Technically, Genet models that conform to the Genet metamodel are networks of generation nodes. Each node provides one XSL transformation of a specified set of sources into target results. As its data sources, generation nodes may use either XML files accessible via URL or XML results of other generation nodes. Each generation node may also store its results to one file or cycle result-set into several files. Gen node result may be either an XML, or any other type of file (i.e. html, pdf, plain text, Java, C++). Genets are used for model transformations and code/artifact generation.

There are the following purposes of the Genet Metamodel:

- Generation of human specific outputs
 - Generation of printable/browsable reports from information stored in models
 - Generation of manuals for software defined in models
- Generation of machine specific outputs
 - Generation of program code in any programming language
 - Generation of any sort of configuration files