



Metada Frontend Modeler 2.1

<http://www.metada.com/>

- [1. Product Information](#) 3
 - [1.1. Executive Summary](#) 3
 - [1.2. Benefits](#) 3
 - [1.3. Product Features](#) 3
 - [1.4. License](#) 4
 - [1.5. Roadmap](#) 5
- [2. Development Process](#) 5
 - [2.1. Modeling](#) 5
 - [2.1.1. Complex Types](#) 5
 - [2.1.2. Services](#) 5
 - [2.1.3. Actions](#) 6
 - [2.1.4. Forms](#) 7
 - [2.1.5. Form Components](#) 7
 - [2.1.6. Exceptions](#) 7
 - [2.2. Validation](#) 7
 - [2.3. Generation](#) 8
 - [2.4. Versioning & Synchronization](#) 8
- [3. Maintenance Procedures](#) 8
 - [3.1. Metarepository Installation Procedure](#) 8
 - [3.2. Modeler Installation Procedure](#) 10
 - [3.3. Create Repository Procedure](#) 11
 - [3.4. Create Model Procedure](#) 11
 - [3.5. LDAP Authentication Setup Procedure](#) 11
 - [3.6. User Group Setup Procedure](#) 12
 - [3.7. User Authorization Setup Procedure](#) 12
 - [3.8. Backup Mechanism Setup Procedure](#) 12
 - [3.9. Disaster Recovery Procedure](#) 12

1. Product Information

1.1. Executive Summary

Enterprises may have tens or even hundreds of applications that are custom-built or acquired from third-party suppliers. The applications may operate on multiple OS platforms and can be geographically dispersed. To effectively manage these applications and to support required business processes, organizations introduce **enterprise systems integration (EAI)** and **service-oriented architecture (SOA)**. In order to fully leverage these technologies Metada Frontend Modeler enables efficient construction of advanced enterprise frontends that provide users access to functionalities offered by their **enterprise service bus (ESB)**.

Metada Frontend Modeler applies model-driven software development approach and enables effective zero-code platform-independent frontend modeling. Frontend models include form definitions, frontend flows, service definitions, service interfaces, and data transformations. With no need for programming, the tool generates configurations and/or complete program code for a selected frontend platform (e.g. **Cleverlance SmartClient**).

Metada Frontend Modeler significantly decreases the time to develop and deploy new frontend components while reducing the cost and time associated with ongoing support and maintenance. It is a proven tool **for frontend analysts** that allows full-scale modeling of organization's **enterprise frontends**. It is designed for complex, heterogeneous IT environments with many users, functionalities, and services.

Metada Frontend Modeler offers:

- modeling tools for platform independent description of enterprise frontends
- model validation
- generation facilities for documentation, program code, and configurations
- model versioning and synchronization

Metada Frontend Modeler is a web-based modeling environment that is always accessible to any number of users with no need to install any software on their local machines.

1.2. Benefits

Fast time to market - Metada Frontend Modeler focuses on ease and speed of delivery of new IT services. This enables enterprises to meet business needs quickly and capitalize on market opportunities.

Efficient management of frontend services - All information about frontend flows, exchanged data structures, and transformations are centrally described, stored, and managed. Information is readily available to all parties and assists in business planning and business process optimization.

Quality - With growing range of validations, Metada Frontend Modeler becomes a highly reliable product for managing enterprise frontend solutions.

Platform Independence - Frontend models are platform-independent and may be executed in various kinds of frontend technologies. It is even possible, during time, to migrate from one platform to another.

Low total cost of ownership - Good management of frontend flows and their comprehensibility by business stakeholders speeds up the development and maintenance of enterprise IT solutions. Platform independence allows for inexpensive transition to new technologies as they are introduced within the enterprise IT architecture.

1.3. Product Features

Metada Frontend Modeler provides several features that enable platform-independent modeling of service-oriented enterprise frontends:

- **Web-based environment**

Any number of frontend analysts may take part on development and maintenance of one central frontend model or even multiple frontend models at the same time. User authentication is either via enterprise LDAP or global OpenID. There is no need to install and maintain any special software on user machines - web-browser is sufficient.

- **Frontend services modeling**

Each user task in an enterprise system is represented by a **flow of actions** that need to be performed. The flow is executed by one particular user during one session with the system. The flow presents user with information and lets the user provide needed information via forms. The flow may include decision actions that take different paths in the flow based on the data being processed. There are also service call actions that may call services on the **enterprise service bus (ESB)** to obtain or provide data.

- **Model validation**

Conformance of frontend models to specific constraints may be regularly checked. Validations ensure production quality and consistence of models. Validations check compliance to conventions or structural constraints that need to be enforced.

- **Model versioning and synchronization**

In enterprise frontend solution it is necessary to support effective long-term development and maintenance. In production model (the one that defines the current state of the frontend), only necessary fixes are usually being realized. Most of the development and changes are performed in development models, which may be migrated to production only after thorough testing. For this reason Metada Frontend Modeler supports multiple versions of models that may be synchronized among one-another.

- **Configuration generation**

Metada Frontend Modeler currently features full configuration generation for **Cleverlance SmartClient**. Code generation for other frontend platforms is possible.

1.4. License

License Conditions

Non-exclusive license is provided for Metada Frontend Modeler, a software product, which is

- Web-based environment for modeling service-oriented enterprise frontends
- Automatic generator of configurations for Cleverlance SmartClient

License does not include any particular frontend model. It is expected that particular frontend models will be defined by frontend analysts (third party, or customer's) using the licensed Metada Frontend Modeler.

License Policy

- Licensed per installation
- One installation within one organization is sufficient
- Unlimited model number and size
- Unlimited users

There are no additional costs for giving access to users from the customer organization or other cooperating organizations. The license is given for one installation and unlimited users of any kind.

Source Code

- The license provides the licensee with full access to the source code.
- Access is provided to the most current version of the source via secured read-only access to the Metada Subversion repository.
- The license allows the licensee to modify the source code in order to improve the current installation, nevertheless use of the source code apart from the licensed installation is strictly prohibited. On the other hand, Metada could not be held liable for any disfunctions that were caused by such additions or modifications.
- Metada obtains full rights to any source code added to the product code-base or modified by the licensee or its contractors.

Metada, at all times, maintains full ownership of all the product source code. The right to add or modify the source code is given only to remove any potential risks that might have occurred if the chance was not given. On the other hand, Metada is ready to make any adjustments needed for the customer's project and it should not be expected that the customer will have to make any changes alone.

Generated Code / New Generators

- The generated code is not a code to be changed by the customer - it is the final code.
- To make changes in what is being generated, generation templates (XSLT templates) may be modified or added as needed.
- Generation networks may be created and modified as needed.

Support & Maintenance Policy

- After subscription to the support & maintenance plan, licensee is entitled to upgrades of the licensed software.
- Upgrades mean improvements of the modeling environment or model management tools (GUI improvements, visual modeling components, model versioning facilities, etc.).
- Upgrades of specific functionalities are not mandatory (if new functionality is required and is not part of the regular upgrading plan, it needs to be requested as a separate development).

Cooperation and Training

- Solution may be operated both on customer's or Metada infrastructure (on customer's server or "in the cloud").
- Installation/maintenance is provided off-site via VPN connection. On-site support also possible.
- Training program for integration analysts is available.

1.5. Roadmap

We have completed the 2.1 release and are beefing up for further improvements in the releases to come.

Note that the list below is subject to change. This is only a high-level map that shows the direction where the development is heading. There might be additional maintenance releases as well.

Metada Frontend Modeler 2.3 (November 2008)

- Revision of structured types modeling (types with dives)
- Revision of form components modeling
- Menu and navigation modeling
- Translator action will enable more advanced mappings without XQuery (XQuery will be generated)
- Online model execution (testing/prototyping)

Metada Frontend Modeler 2.4 (June 2009)

- Merge actions into services
- New graphical modeling of forms and flows
- Revision of forms modeling (form fragment reuse, similar to types structuring)
- Model autotesting tools selection

Metada Frontend Modeler 2.5 (November 2009)

- Online generation and browsing of documentation
- Extended prototyping functionalities
- Extended mappings
- Import/export of service interface XSDs
- Authorization modeling

2. Development Process

2.1. Modeling

2.1.1. Complex Types

Complex types are used to define structures of service interfaces. Several complex types may be linked to a service as its input, output, or both. Complex type has:

- **Name** - Name of the XML element for this complex type. This name is used only for items in a collection, because otherwise complex types are named at the place where they are used (i.e. in service parameters or complex type properties).
- **Collection name** - Name of the XML element used to wrap collection of the complex type. Not used - since name to collection is given in service parameter or complex type property that links to the given complex type.
- **Supertype** - Defines another complex type that is extended by the given complex type.

Complex type itself is defined as a set of properties of three possible kinds:

- **Element property** - Defines a property with a given basic data type (boolean, char, date, float, int, string, timestamp, url) that should be represented in XML as an element with a value.
- **Attribute property** - Defines a property with a given basic data type that should be an XML attribute.
- **Complex type property** - Defines reference to another complex type that should become a substructure of the given complex type.

Each complex type property has the following further settings:

- **Name** - Defines element/attribute name.
- **Collection** - Defines whether the property is a collection (i.e. the value or substructure has more than one occurrence).
- **Composite** - Defines whether the substructure is owned by the complex type that includes it.
- **Default value** - A value of the property that should be used when the property is serialized and has no value.

At the moment, complex types are used only in the model to declare service interfaces, nevertheless this information is used only to offer flow context mappings, but is not used anywhere in generated configurations.

2.1.2. Services

A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services. Services are used either by human users or machines to perform some specific function in a software system. Frontend modeler recognizes the following types of services:

- **Frontend** - Frontend service defines a task flow executed by a user. The flow contains human tasks (screens/forms) that the user is expected to react to by entering data and actively progressing the flow. The flow itself may make decisions based on its data context and call other services.
- **Integration** - Integration service defines an orchestration flow (microflow) where other services are called and decisions based on processed data may be made. There are no human tasks in the flow and therefore the flow may be executed as part of an ESB (Enterprise Service Bus).
- **Backend** - Backend service represents a specific service offered by some external system. No flow is defined for this type of service.

Each service may have the following attributes defined:

- **Name** - Technical name of the service recognized by the runtime (i.e. the real name under which the service may be technically called).
- **Business name** - Name under which human users recognize the service.
- **First action** - (Frontend and integration services only) First action in the service flow.
- **Communication type** - (Backend services only) Type of technical communication channel in which the service is available. Specific channels depend on the runtime architecture.
- **Service URL** - (Backend services only) URL under which the service is available (for http communication type).

Service interface is defined by a set of service parameters. Each service parameter defines:

- **Direction** [in|out|inout] - Specifies if the service accepts the parameter as input (in), as output (out), or both (inout).
- **Name** - Name of the element that contains the value or structure.
- **Type** [element|complex] - Parameter either directly represents an element with a value or a link to complex type.
- **Data type** [boolean|char|date|float|int|string|timestamp|url] - Basic data type of element parameter.
- **In collection** - Specifies if there is a collection of the element or complex type.
- **Required** - Specifies whether the parameter is required.
- **Position** - Relative ordering of parameters.

Several exceptions may be linked to a service. Service exceptions may be handled in a flow that calls such a service.

2.1.3. Actions

Actions define individual steps in service flows. One action is only part of one service flow. There are the following types of action:

- **Branch** - Branch action is a decision that selects one branch that should be followed out of several branches possible. There is a condition expression for each branch evaluated.
- **Form** - Form action represents a user task, where the user may be presented data and may be asked to provide data to the flow.
- **Service** - Service action represents a call to another service.
- **Transform** - Transform action creates new data in the flow context by transformation of the existing context.
- **End** - End action defines end of the flow.

Each action has the following attributes defined:

- **Service** - Action maybe linked only to one given service.
- **Name** - Technical name of the action which is used to address data the action produced within context.
- **Business name** - Name of the action used for documentation purposes.
- **Terminator** - Specifies if the flow may be terminated in the given action.
- **Form redisplay** - Specifies if the form should be completely reinitiated from input data in case that it is called several times in one particular flow instance (otherwise, form will appear with the data that were entered in the form during its previous appearance).

Each **form or service action** may have several parameters defined. Parameters are used to map data from context to the form or service. Each action parameter defines:

- **Source path** - Path in the context where data should be obtained.
- **Target path** - Path in the target to which the value should be mapped.
- **Position** - Relative position of the parameter.

Each **transform action** may have several XQuery transformation defined that execute the XQuery against the flow context and write the result into specified parameters in the context.

Each **branch action or service action** may have several branches with conditions specified. Each branch definition includes:

- **Business name** - Name of the branch for documentation purposes.
- **Branching condition** - XPath expression executed against the flow context (with boolean result). If evaluated to true, the branch is taken (the action specified for the branch is called).
- **Exception** - Exception may be used as a branching condition in case that the called service has exceptions defined.
- **Next action** - Action that should follow if the condition satisfied.
- **Exception to throw** - Exception to be thrown in case that the condition is satisfied.
- **Exception for user** - Specifies if the thrown exception is shown to the user.

- **Form redisplay** - Fully redisplay the form that follows this branch.

2.1.4. Forms

Forms are used to present information to the user or obtain it from the user. Forms are linked to form actions of frontend service flows. Each form has a technical name used to identify the form in runtime and business name used for documentation.

Each form is composed of **form fields**. Each form field contains:

- **Name** - Technical name of the field (used to address values in the form).
- **Type** - Widget type of the field (button, calendar, combobox, component, checkbox, groupbox, image, label, listbox, radiobutton, textarea, textbox, table).
- **X, Y, Z, width, height** - Position and size of the field.
- **Label placement** - Direction from the field where the field label is placed (W, N, NW, L, UL).
- **Visible** - Specifies if the field is visible.
- **Enabled** - Specifies if the field is enabled (may be changed by the user).
- **Default value** - Value that should appear in the field if no value comes in data.
- **Image URL** - If type image, this defines the image url.
- **Group** - Radio button group identifier.
- **Tab index** - Numbers specifying order in which user may move from one field to another by pressing a tab key.
- **Style** - CSS style class to be used to style the field.
- **Form component** - Form component that should be placed on a form (for field type component).
- **Active** - Only active fields are used on the form.

Each form field may have a **static enumeration of values** assigned. The list of values is shown in the form field (usually listbox or combobox) and the user has to select from the provided values.

For **table form field** it is possible to define **table columns**. Each table column is specified with:

- **Name** - Technical name for the column (used to address values in the column).
- **Type** - Widget type (static, submit, textbox, combobox, checkbox).
- **Width** - Column width in pixels.
- **Send output** - Specifies if contents of this column should be written in data provided by the form.
- **Hidden** - Specifies if the column is hidden.
- **Position** - Relative ordering of columns.
- **Style** - CSS style class to be used to style the column.

Each form field and table column may have **validations** defined. Validation definition contains:

- **Mandatory** - Field is required to have a value.
- **Min length** - Minimum length of field value.
- **Max length** - Maximum length of field value.
- **Mask** - Entry mask of the field (will let user enter only specific characters).
- **Regexp** - Regular expression that has to be satisfied.
- **Min value** - Minimum integer value.
- **Max value** - Maximum integer value.

2.1.5. Form Components

Form components are used to implement very specific functionalities within a form. A form component is an implemented form widget which is identified in the model and may be linked to forms as a component form field.

2.1.6. Exceptions

Exceptions are thrown by services or service actions of type branch. Exception may be presented to the user directly, or it may be handled by an exception handling part of a service flow. Exceptions have the following attributes:

- **Code** - Exception code used to identify the exception in the runtime.
- **Name** - Technical name of the exception.
- **Type** - (business, technical).

2.2. Validation

Each frontend model may be validated for consistence and completeness. The following validations are being performed:

Forms

- Form name must be unique

- Field names must be unique

Services

- Service name must be unique
- Frontend service must have first action

Actions

- Form action must have form assigned
- Form action must have next action assigned
- Service action must have service assigned
- Service action must have next action assigned
- Branch action has to have at least one branch

Exceptions

- Exception code must be unique
- Exception must have business description

2.3. Generation

Metada Frontend Modeler is a model-driven software development environment focused on platform-independent modeling of frontend systems. The models created in the modeler have no specific settings for any frontend platform, nevertheless the specifics may be added via code generation for the specific platform. When model is valid (i.e. there are no validation errors), it is possible to proceed with code/configuration generation. For each frontend model, Metada Frontend Modeler contains a generation page which offers generation buttons.

Frontend modeler generates configurations that are interpreted by an external runtime framework (e.g. Cleverlance SmartClient). There are the following configurations generated:

- **Forms** - For each form in the model there is one XML file generated that contains complete form definition.
- **Frontend flows** - For each frontend service defined in the model there is one XML file generated that contains complete frontend flow including all actions and definitions of called services.
- **Exceptions** - For all exceptions defined in the model there is one XML file generated that contains descriptions of all exceptions.

2.4. Versioning & Synchronization

When a frontend model is completed and migrated into production (i.e. starts to be used to run frontends), a next round of modeling may start for the next release. Metada Frontend Modeler allows management of several concurrent versions of the same frontend model. One version of model may be in production, another in testing, and yet another in development. Usually it is sufficient to have two concurrent versions:

- **Version in production** - A version of a frontend model that is the one used in production. Minor fixes are being done to this version in cases that errors are encountered in production environment.
- **Version in development** - A future version of the model that will one day become the version in production - one that is being developed and tested.

Compare & Synchronize Functionality

When several concurrent versions of a model are managed it is necessary to synchronize changes in one version to another (e.g. to propagate the fixes done in production into the version in development). Metada Frontend Modeler provides compare & synchronize functionality that enables arbitrary models to be compared between each other, and enables selected changes to be propagated from one model to another. Any two models may be compared and synchronized.

3. Maintenance Procedures

3.1. Metarepository Installation Procedure

This procedure defines steps that need to be taken to install Metada Metarepository 2.1.

Context

Metada delivers various modeling tools and metamodels, such as Integration Modeler or Frontend Modeler. These modelers share Metada Metarepository as its runtime infrastructure. Therefore, in order to use these additional tools, it is necessary to install the Metarepository first.

Procedure steps

1. Prepare environment - hardware

Prepare server hardware with at least the following specification:

- 2.8 GHz processor
- 2 GB memory
- 60 GB storage
- network card

2. Prepare environment - software

Pre-install and configure the following software:

- Windows Server 2000 or higher
- Java Development Kit (JDK) 6 or higher (<http://java.sun.com/>)
- Subversion Version Control System (<http://subversion.tigris.org/>) - Subversion is used for storage and versioning of models. Note that this software may be located on a different server and used remotely.
- Jakarta Tomcat 5.5 server or higher (<http://tomcat.apache.org/>) - Jakarta Tomcat or similar servlet container is necessary to run the Metarepository web application.
- Apache HTTP Server 2.0 or higher (<http://httpd.apache.org/>) - Apache HTTP Server is useful for secured connections (https) configuration and WEBDAV access to the Subversion VCS.

3. Download and placement

Two web application archives (wars) are provided:

- mambomde.war (<http://www.metarepository.com/downloads/2.1/mambomde.war>)
- openid.war (<http://www.metarepository.com/downloads/2.1/openid.war>)

Download the two war files from the above locations and place them into your servlet container (e.g. the webapps directory in Tomcat).

4. Configuration

There is one configuration file for the web application: "config.xml". The configuration file must be placed in the application root directory. The root directory may be specified by JVM system property "mambomde.rootdir" (default value is "c:/metada/mambomde").

Contents of the config.xml file follow:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--
  Offline mode is default. Online mode is turned on by adding offline="false".
  If an installation is in offline mode any username is always authenticated
  (password is not checked)! Reason for this is that if someone is able to
  setup new or change mode of existing installation to offline mode. She
  already has full control over this installation is it's security is already
  compromised.
-->
<MamboMDEConfig offline="false">
  <!--
    Specifies which database XDA uses to store data. Available types are:
    "msaccess" and "mysql".
  -->
  <XDA dbtype="msaccess"/>
  <!--
    LDAP authentication settings.
  <AuthService>
    <LDAPSettings url="ldap://example.com:636/">
      <Principal>cn=%ldapuser%,ou=Users,dc=usr,dc=example,dc=com</Principal>
    </LDAPSettings>
  </AuthService>
  -->
  <!--
    Proxy selector configuration. Other way of configuring http proxy is
    to pass to JVM these command line parameters: http.proxyHost, http.proxyPort
    and http.nonProxyHosts (this Proxy selector should not be used then)
  <HttpProxy host="localhost" port="8080" nonproxyhosts="*.example.com;localhost;127.0.0.1"/>
  -->
  <!--
    Configures logging subsystem (log4j). Format of this element value is
    the same as of any log4j configuration properties file.
  -->
```

```
-->
<Logging><![CDATA[
log4j.rootLogger=WARN, console
log4j.logger.com.metada=DEBUG, mambomdefile
log4j.additivity.com.metada=false
log4j.logger.com.mambomde=DEBUG, mambomdefile
log4j.additivity.com.mambomde=false
log4j.logger.org.springframework=WARN, mambomdefile
log4j.additivity.org.springframework=false
log4j.logger.uk.ltd.getahead.dwr=WARN, mambomdefile
log4j.additivity.uk.ltd.getahead.dwr=false
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d [%t] %-5p %c - %m%n
log4j.appender.mambomdefile=biz.minaret.log4j.DatedFileAppender
log4j.appender.mambomdefile.layout=org.apache.log4j.PatternLayout
log4j.appender.mambomdefile.layout.ConversionPattern=%d [%t] %-5p %c - %m%n
log4j.appender.mambomdefile.Directory=c:/metada/mambomde/logs
log4j.appender.mambomdefile.Prefix=mambomde.
log4j.appender.mambomdefile.Suffix=.log
log4j.appender.mambomdefile.Append=true
]]>
</Logging>

</MamboMDEConfig>
```

5. Run

Run the servlet container (e.g. startup a Tomcat server) and navigate to [http://\[server\]/mambomde/](http://[server]/mambomde/). Metarepository start page appears with no repositories loaded.

6. Load metarepository models

In the "Load Existing Repository" box enter "<https://svn.metada.com/m21/rep>" as the repository url and press the "load" button (no login needed). The loaded repository contains metamodels needed to create other repositories and models.

After loading the Metarepository models the installation procedure is completed. Please, see further procedures for creating new repository or loading modeler specific repositories (such as Metada Integration Modeler repository).

In case that an update to m21 repository is provided, it is possible to make an SVN update. In such a case navigate to "M2.1 Repository" in the top menu, then go to "Model Control" in the left menu, select models to update in the list of models, and finally press the "SVN Update" button.

3.2. Modeler Installation Procedure

This procedure specifies how to load Metada Frontend Modeler 2.1 into already installed Metada Metarepository.

Context

Metada Frontend Modeler uses Metada Metarepository as its runtime environment, where it runs as a repository that contains a frontend metamodel (fm_mta_v2_1) and generation network (fm_genet_v2_1).

Procedure steps

1. In running Metarepository instance navigate to [http://\[server\]/mambomde/](http://[server]/mambomde/) page.
2. In the "Load Existing Repository" box enter "<https://svn.metada.com/fm/rep>" as the repository url and press the "load" button. Login and password is required (provided as part of the license or the support and maintenance agreement).
3. When loaded, the repository appears in the repositories box as "fm_rep". Navigate to this repository.
4. Navigate to the "FM Repository" category in the top menu and then to "Model Control" in the left submenu.
5. Select fm_mta_v2_1 and fm_genet_v2_1 models, and press the "SVN Checkout" button.

After successful checkout of the metamodel and generation network, the Metada Frontend Modeler is ready to be used. At this time it is possible to load existing repository with frontend models or create a new one (please, see further procedures).

The metamodel and genet may be updated as needed from the Model Control by selecting the models and pressing "SVN Update".

3.3. Create Repository Procedure

This procedure specifies how to create new repository where frontend models will be stored and versioned.

Context

Metada Frontend Modeler supports multiple repositories with frontend models. Each repository is versioned in Subversion version control system.

Procedure steps

1. Create new Subversion repository in your Subversion instance (e.g. "mycompany").
2. In the created repository create new "rep" directory.
3. Check, that this repository is accessible via browser by navigating to url of the Subversion instance (e.g. [https://\[server\]/svn/mycompany/rep](https://[server]/svn/mycompany/rep)).
4. In running Metarepository instance navigate to [http://\[server\]/mambomde/](http://[server]/mambomde/) page. Make sure that "fm_rep" is loaded (see Modeler Installation Procedure).
5. In the "Load Existing Repository" box enter the repository url (e.g. [https://\[server\]/svn/mycompany/rep](https://[server]/svn/mycompany/rep)) and press the "load" button.
6. When loaded, the repository appears in the repositories box as "mycompany_rep". Navigate to this repository.
7. Go to "Model Control" on the right side of the menu bar.
8. Go to "User Groups" and create user groups with users that will need access to the repository.
9. Go to "Systems" and create system "rep" and set R to "all" and RW to an administrators group.

After this procedure your new repository will be ready for models to be created (see further procedures).

3.4. Create Model Procedure

This procedure specifies how to create new frontend model in existing frontend repository.

Context

Metada Frontend Modeler supports multiple repositories with frontend models. Each repository is versioned in Subversion version control system.

Procedure steps

1. Create new directory in your Subversion repository (e.g. "test").
2. Check, that this repository is accessible via browser by navigating to url of the Subversion instance (e.g. [https://\[server\]/svn/mycompany/test](https://[server]/svn/mycompany/test)).
3. In running Metarepository instance navigate to [http://\[server\]/mambomde/](http://[server]/mambomde/) page. Go to your company repository (e.g. "mycompany_rep").
4. Go to "Model Control" on the right side of the menu bar.
5. In the "Create New Model" box type model name (e.g. "test") and metamodel name "fm_mta_v2_1", and press create.
6. The new "test" model appears in the list of created models.
7. To synchronize the new model with Subversion, select the model and press the "SVN Checkout" button.
8. Go to "System" in the left menu and edit the new "test" model - change authorization to groups that should have access to the model.
9. Go to "Menu" in the left menu and create "root" menu category. Then create "test" menu category with model URL "test". In the root menu category place the test category as a subcategory. Write descriptions of the categories.

After this procedure your new frontend model is ready for modeling and is accessible from the repository menu.

3.5. LDAP Authentication Setup Procedure

In case that there is a company active directory or LDAP server, it is possible to use it for user authentication to the Metada Metarepository. This procedure specifies how to configure the product to access the identity server.

Procedure steps

1. Obtain LDAP url (e.g. <ldap://example.com:636/>)
2. Obtain LDAP principals for the users (e.g. cn=%ldapuser%,cn=Users,dc=division,dc=example,dc=com)
3. Open C:\metada\mambomde\config.xml file
4. Uncomment the <authservice> element and configure LDAP, for example:

```
<LDAPSettings url="ldap://example.com:636/">
  <Principal>cn=%ldapuser%,cn=Users,dc=division,dc=example,dc=com</Principal>
  <Principal>cn=%ldapuser%,cn=ExternalUsers,dc=division,dc=example,dc=com</Principal>
</LDAPSettings>
```

5. Save the configuration file and restart the "Metada Metarepository" Windows service

3.6. User Group Setup Procedure

This procedure defines how to create a new user group and authorize it to specific models.

Context

It is good practice not to assign users directly to models and authorize them only via user groups. Access to individual models is then provided to users through user groups they are assigned into. A user group is a collection of users with the same needs and privileges. If a user group is assigned to a model, all users in this group get the same level of access. There are the following access levels for each model:

- **Read** - User may only read model contents.
- **Read-Write** - User may read and write model contents.

Procedure steps

1. Navigate to the Model Control and then to User Groups menu
2. Click on "Create" button under the list of existing user groups
3. Define "Name (code)" for the new user group and add at least one existing User ID to "Members" field.
4. Press "Create" button create the new user group
5. Add user group to specific models either with Read or Read-Write privilege

3.7. User Authorization Setup Procedure

This procedure specifies how to give specific users access (authorization) to specific models.

Context

Authentication is provided by OpenID or LDAP, and therefore any user may log into the application, nevertheless to see any models, users have to be authorized to see them. Users are not assigned to models directly, but are assigned to user groups. User groups are then linked to models either to have read-only or read-write access.

Procedure steps

1. In Model Control navigate to "User Groups" section
2. Add user to an existing user group or create a new user group (see User Group Setup Procedure)

3.8. Backup Mechanism Setup Procedure

This procedure specifies how to setup mechanism for regular automatic storing of models.

Context

Backups are a way to protect the investment in the modeling efforts. Regular backup to different secured locations ensures that models will be recoverable even in case of hardware failures, software bugs, viruses, human actions, or natural disasters. The most valuable assets created in Metada Metarepository are integration models. These models should be regularly checked-into version control (Subversion). What needs to be backed-up then are the version control data.

Procedure steps

1. Create operating system job that runs regularly (e.g. once a day at night)
2. Configure the job to check-in all changes in all models
3. Configure the job to pack and copy the whole Subversion repository to a different location (e.g. Amazon S3)

3.9. Disaster Recovery Procedure

This procedure specifies what to do in case of complete or partial hardware or software failure that causes loss of integration models or Metada Metarepository disfunction.

Procedure steps

1. In case of complete destruction - first execute the installation procedures
2. In case of Subversion repository destruction, load and recover the full repository from the latest backup
3. Reload the models from Subversion repository