



for efficient and iterative
development of smart factories



ITEA 3 – 15015

Work package 3
Creation and Distribution of Component Models



Deliverable D3.2
Model distribution system specification

Document type	: Deliverable
Document version	: 1.0
Document Preparation Date	: 2018-06-30
Classification	: public
Contract Start Date	: 01.09.2016
Contract End Date	: 31.08.2019



	<p style="text-align: center;"> ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015 Project Coordinator: Thomas Bär, Daimler AG </p>	
---	---	---

Final approval	Name	Partner
Review Task Level	Mario Thron	IFAK e.V.
Review WP Level	Christian König	TWT GmbH
Review Board Level	Manuel Paul	FESTO AG & Co.KG

	<p style="text-align: center;">ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	--	---

Executive Summary

This document provides a specification of a Model Store, which is necessary for exchange of engineering models between different companies. Those models are related to mechatronic components, which are assembled to production systems. The concept of packages has been introduced, while a package can contain multiple models as files (e.g. geometry models and behavior models), which are described within a package description file. The Model Store is designed as client-server application. The client specification requires the management of package dependencies if e.g. other component models are aggregated to a system model.

Contents

Executive Summary	3
1 Introduction.....	6
2 Use Case Analysis and Requirements.....	7
2.1 User Roles and Systems and Use Case Overview	7
2.2 Use Case Descriptions	9
2.2.1 Management of Multiple Model Store Servers on Client Side.....	9
2.2.2 Management of Distributed Hosting.....	9
2.2.3 Downloading of Models with Dependencies	10
2.2.4 Search of Component Models.....	11
2.2.5 Requesting Costs of Models	11
2.2.6 Payment Scenario.....	12
2.2.7 Access Token Handling	13
2.2.8 Uploading of Models to a Model Store	14
2.2.9 Financial processes.....	14
2.2.10 Account and Access Management	15
2.2.11 Model Storage on Server Side	16
2.3 Requirements resulting from the Use Cases	17
3 System Design	19
3.1 Initial Design Decisions.....	19
3.2 Decomposition of Model Store Server and Client.....	19
3.2.1 Client CLI.....	20
3.2.2 Client GUI.....	22
3.2.3 Client Config	23
3.2.4 Package Store	23
3.2.5 Package Description Cache.....	24
3.2.6 Client Adapter.....	25
3.2.7 Server Adapter.....	25
3.2.8 Server Config	26
3.2.9 Admin CLI.....	28
3.2.10 Payment Service	28
4 Summary and Outlook.....	30
References.....	31

Annex A: Formal Description of the Model Store Server Interface 32

Annex B: Package Description File Format..... 38

List of Figures

Figure 1: Terminology used in this document as UML class diagram 6

Figure 2: Systems and related user roles 7

Figure 3: System architecture of the Model Store 19

Figure 4: Structure of a configuration file for an ENTOC ModelStore Client 20

Figure 5: Draft design of the Client GUI 22

Figure 6: The Folder Structure of the Package Store 24

Figure 7: General structure of the Model Store Server configuration file. 27

Figure 8: User roles within the Model Store Server configuration file 27

Figure 9: Example for a payment data store 29

1 Introduction

The purpose of the ENTOC project is to provide a tool chain of engineering tools, which makes it possible to re-use data of previous engineering steps within the current engineering step. This makes it possible to harmonize data structures, their semantics and their encoding. A further challenge is to provide places, where to store and acquire necessary data. Within this document we specified a system, which we call Model Store. It is specified according to the client and server architecture pattern, which easily makes it possible to access the engineering data by different engineers.

A basic assumption is that a production system can be split into mechatronic components, such as robots, conveyors or others. Engineering data, related to those components, can be of different nature, like geometry models, behavior models and interlinking component description models. We introduce the term "package" as a combination of models and a package description. The package structure is presented as an example in [D3.1] and formally described within this document (see Annex B). Thus the Model Store is the main utility to manage those packages containing engineering information about mechatronic components (see Figure 1). It is important to note, that the provider of models upload the models as complete packages, while it should be possible to download only selected model types of the package (so parts of a package only).

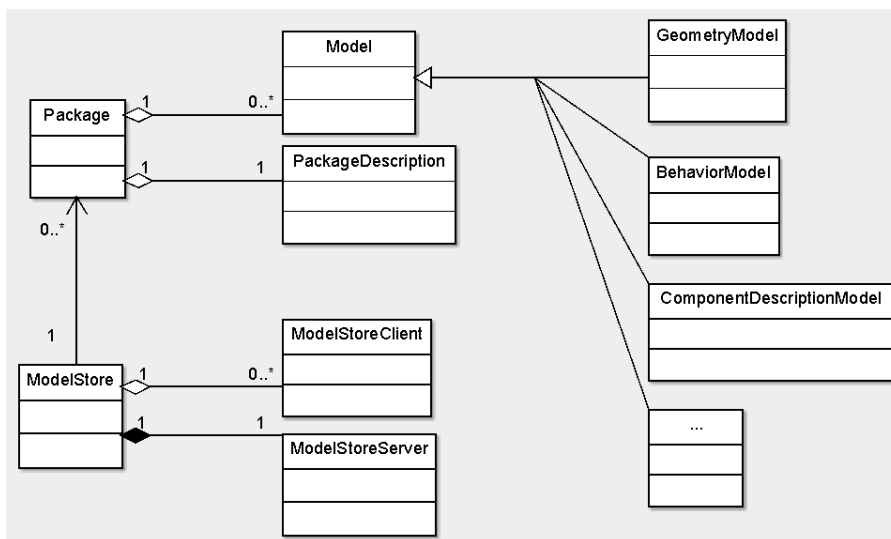


Figure 1: Terminology used in this document as UML class diagram

A typical usage scenario would be that a provider of a mechatronic component provides a package containing a CAD-based geometry model and a behavior model. The geometry model can be used by planners who assemble different components to a system view for the system integrator, such as a company designing the production plant. The behavior model can be used within the virtual commissioning phase of the project. So data supply and usage are at different time and the data is provided and used by different engineering teams.

The intention of this document is to provide initially a detailed use-case analysis (chapter 2), a list of requirements (section 2.3) and system design specification (chapter 3). That specification provides information about data structures and interfaces between tools. The annex of this document contains formal specifications of the package description file structure as well as a formal specification of the interface of the Model Store Server, which makes it possible to provide conformant implementations of that server.

2 Use Case Analysis and Requirements

The specification of the Model Store is approached in a use-case driven fashion. This chapter describes the use cases for the proposed Model Store. First, three different roles and three systems and their relations are described. In the next step, use-cases for the user roles and systems are drafted, to create requirements based on these use-cases.

2.1 User Roles and Systems and Use Case Overview

The approach presented in the following chapters is a combination of the approach for distribution of software packages for Open Source operating systems (like Linux, NetBSD or FreeBSD) and the commercial approach used for distribution of so called Apps for mobile devices by using central App repositories (like Apple App Store or Google Play Store).

Software packages for operating systems should be efficient in terms of re-using existing code and in terms of disk space. Therefore multiple binaries of an operating system share the use of pre-compiled software libraries and tools. Thus a software package depends on those libraries and tools. If the software package is to be installed from a package repository, then all libraries and tools, this software package depends on, have to be checked and installed if necessary.

Engineering is the process of creating models of products or production systems and also this process has to be managed in an efficient manner. Re-using of partial models is an appropriate approach for that purpose. If a larger system model is composed of smaller component models then it should be possible to download the components from a repository too. But if this system model later on will be integrated into a super-system model, then the dependent component models have to be integrated too, which implies, that the system and component models have to be available from a repository, which we call a Model Store in the ENTOC project.

The commercial aspect of the Model Store is similar to the commercial model used by mobile device App repositories. Multiple vendors create component models and upload it to the Model Store. A Model Store operator manages the distribution of the models to clients and is paid from that collected money, but some money is retained by the Model Store operator for his model distribution services.

Figure 2 depicts elements and user roles in context of the Model Store. The main system for handling the data will be the Model Store Client, which provides access to the server for any human user.

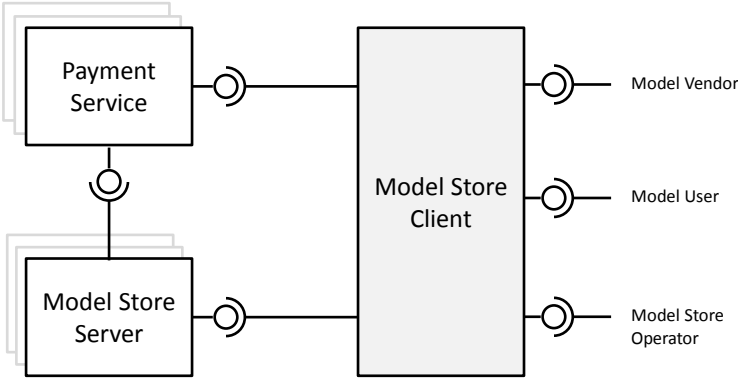




Figure 2: Systems and related user roles

	<p>ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

The following table describes systems and user roles in detail:

No.	<i>User Role or System Name</i> and its Description
1	<p><i>Model User</i></p> <p>A <i>Model User</i> is a user role, which is able to download, search for and retrieve meta information about packages from the vendor sections in a <i>Model Store Server</i> it has granted access to. Furthermore, it is able to search for, retrieve meta information, update and delete packages from its local <i>Model Store Client</i>.</p>
2	<p><i>Model Vendor</i></p> <p>A <i>Model Vendor</i> is a user role, which provides Packages. It is capable of all operations a <i>Model User</i> can carry out. In addition, it is able to upload new packages to its own vendor section on a <i>Model Store Server</i> including new versions of older packages.</p>
3	<p><i>Model Store Operator</i></p> <p>A <i>Model Store Operator</i> is a user role, which is capable of all operations of a <i>Model Vendor</i> . It has access rights to all vendor sections on its related <i>Model Store Server</i> and can manage the users and their roles on its related <i>Model Store Server</i>.</p>
4	<p><i>Model Store Server</i></p> <p>A <i>Model Store Server</i> is a system, which stores all packages from <i>Model Vendors</i> who have access to it in the structure described in section 3.2.4. It provides all necessary APIs to perform operation on it. There can be multiple <i>Model Store Servers</i> in a distributed scenario.</p>
5	<p><i>Model Store Client</i></p> <p>A <i>Model Store Client</i> is a system, which runs on a local environment. It is capable of establishing a connection to a <i>Model Store Server</i> (sending requests and getting responses), processing responses from a <i>Model Store Server</i>, managing a local Model Store Storage and perform operations either on the local Model Store Storage or on a <i>Model Store Server</i> via API calls to it.</p>
6	<p><i>Payment Service</i></p> <p>A <i>Payment Service</i> is a system, which regulates the payment processes of the Model Store applications. Existing Payment Services are the on-line services from PayPal, VISA, Mastercard or other. Model Users pay via this service onto the "bank account" of the Model Store Operator. The collected money is later on distributed among Model Vendors and the Model Store Operator.</p>

2.2 Use Case Descriptions

The use cases are described by the user or system that has his use case (main perspective), the goals that shall be achieved by the use case, and the typical steps that are executed to achieve this goal (standard workflow). From these use cases, requirements are derived that are described in further detail in section 2.3.



Some of the use cases are considered necessary for the basic functioning of the Model Store. Those use cases that relate to the Payment Service are considered optional in this specification.

2.2.1 Management of Multiple Model Store Servers on Client Side

Name	Management of Multiple Model Store Servers on Client Side
Main Perspective	<i>Model User, Model Vendor</i>
Goals	Any user of the <i>Model Store Client</i> can configure an arbitrary number of <i>Model Store Servers</i> that will be queried when a model has to be downloaded. The configuration includes a prioritization of the <i>Model Store Servers</i> .
Standard Workflow	The Model User configures the Model Store Client via configuration file or commands to use different Model Store Servers in a specific order. Such commands would be <code>add_server</code> or <code>up/down_server</code> (for prioritization).
Comments	References to models don't include information about the location of the model on a specific <i>Model Store Server</i> . This improves the flexibility when models are exchanged between different companies. In order to resolve model references it is necessary to provide <i>Model Vendor</i> , model name and version id, the <i>Model Store Client</i> has to be configured for the specific <i>Model Store Servers</i> to be used.
Resulting Requirements	<i>Multiple Model Store Servers in Model Store Client</i>

2.2.2 Management of Distributed Hosting



Name	Management of Distributed Hosting
Main Perspective	<i>Model Store Server, Model Store Operator</i>
Goals	A <i>Model Store Server</i> can be configured to proxy model requests to other <i>Model Store Servers</i> . Model data stored on the remote <i>Model Store Server</i> can be cached in the local <i>Model Store Server</i> . If the remote <i>Model Store Server</i> requires authentication, the local <i>Model Store Server</i> can be configured to use specific login information for the remote <i>Model Store Server</i> by the <i>Model Store Operator</i> .

	<p>ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

Standard Workflow	<ul style="list-style-type: none"> • The <i>Model Store Operator</i> logs in into the <i>Model Store Server</i>. • The <i>Model Store Operator</i> adds a remote <i>Model Store Server</i> as a separate repository. • The <i>Model Store Operator</i> provides login information for remote <i>Model Store Server</i> to be used instead of the user's login information when accessing model information through this repository.
Comments	This use case will not be applied when a centralized <i>Model Store Server</i> is used.
Resulting Requirements	<p><i>Model Store Payment</i></p> <p>The <i>Model Store Server</i> SHOULD be able to support payment</p>

2.2.3 Downloading of Models with Dependencies

Name	Downloading of Models with Dependencies
Main Perspective	<i>Model User</i>
Goals	The <i>Model User</i> has downloaded one or more models. There are transactions per single model. The <i>Model User</i> is informed, whether all requested models have been downloaded or which of the models could not be downloaded (and the failure cause). If there are dependencies of a model requested by the <i>Model User</i> then all depending models are downloaded too.
Standard Workflow	<ul style="list-style-type: none"> • The <i>Model User</i> defines the list of models to be downloaded. • The <i>Model User</i> calls a command for downloading at the <i>Model Store Client</i> application and uses the list of models as command parameter. • The <i>Model Store Client</i> calculates a list of dependent models to be downloaded too and asks the <i>Model User</i> to acknowledge the complete model list (or abort of the operation). • The <i>Model Store Client</i> then downloads each of the models from the <i>Model Store Server</i> in a transactional manner. • The <i>Model Store Client</i> informs the <i>Model User</i> about success or failures of the operation.
Comments	The <i>Model Store Client</i> is responsible for the dependency management.
Resulting Requirements	<p><i>Store components data</i></p> <p><i>Store component metadata</i></p>



	<p>ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

2.2.4 Search of Component Models

Name	Search of Component Models
Main Perspective	<i>Model User</i>
Goals	<p>The <i>Model User</i> gets a list of short descriptions of models according to a query. The query may involve one or more elements:</p> <ul style="list-style-type: none"> • A <i>Model Vendor</i> name search expression • A component model name search expression and • A model type like behavior model or symbol model • A version search expression. <p>Search expressions may include unsharp definitions, like regular expressions for strings or minimum version numbers.</p>
Standard Workflow	<ul style="list-style-type: none"> • The <i>Model User</i> defines the query. • The <i>Model User</i> calls a command for searching models at the <i>Model Store Client</i> application and uses the query as command parameter. • The <i>Model Store Client</i> sends the query to the server, which determines matching models and returns a list of models and their according package descriptions. • The <i>Model Store Client</i> presents the resulting list of models to the <i>Model User</i>.
Comments	The model type will be a property within the package description files.
Resulting Requirements	<i>Searching for components</i>

2.2.5 Requesting Costs of Models



Name	Requesting Costs of Models
Main Perspective	<i>Model User</i>
Goals	The <i>Model User</i> is informed about the costs for downloading a set of models and depending models.
Standard Workflow	<p>This workflow is embedded into the workflow of downloading models.</p> <p>When the <i>Model Store Client</i> calculates the list of dependent models and shows the information about the models, then cost information should be included as information per model and as summary information about all models to be downloaded.</p>

	<p>ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

	<p>It is part of further specifications, whether the costs can be calculated on client side or whether the <i>Model Store Server</i> must be queried each time, when a download process is initiated. (see "Downloading of Models with Dependencies")</p>
Comments	<p>This is an optional feature of the <i>Model Store Client</i> and <i>Model Store Server</i>.</p>
Resulting Requirements	<p><i>Multiple Model Store Servers in Model Store Client</i></p>

2.2.6 Payment Scenario



Name	Payment Scenario
Main Perspective	<i>Model User</i>
Goals	<p>The <i>Model User</i> has paid for the download of models and has got an invoice. The money is now managed by the <i>Model Store Operator</i>.</p>
Standard Workflow	<p>This workflow is embedded into the workflow of downloading models. The following sub-workflow has to be performed before the <i>Model Store Client</i> downloads each of the models:</p> <ul style="list-style-type: none"> • The <i>Model Store Client</i> sends the list of models to be downloaded and the expected summary cost to the <i>Model Store Server</i>, which calculates the summary costs at the server side. If there are deviations between the two calculated costs, then the <i>Model Store Server</i> sends a message to the client, that the model meta-information is outdated. The <i>Model Store Client</i> then asks the <i>Model User</i> to update the model meta-information. • The <i>Model Store Server</i> requests at the <i>Payment Service</i> whether the <i>Model User</i> (or the respective company) is able to pay for all models. If so, then the <i>Model Store Server</i> signals it to the <i>Model Store Client</i>, which then performs the download of the models in a transactional manner. • After each model download transaction a log entry is stored at the <i>Model Store Server</i>. • The <i>Model Store Server</i> sums up the costs for each downloaded model and after a certain delay books the money at the <i>Payment Service</i>. <p>(see use case 2.2.3 "Downloading of Models with Dependencies")</p>
Comments	<p>This is an optional feature of the <i>Model Store Client</i> and <i>Model Store Server</i>.</p>

	<p style="text-align: center;"> ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015 </p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

Resulting Requirements	<i>Multiple Model Store Servers in Model Store Client</i>
------------------------	---

2.2.7 Access Token Handling

Name	Access Token Handling
Main Perspective	<i>Model User</i>
Goals	A <i>Model User</i> creates or removes an access token for his account on the <i>Model Store Server</i> . This access token can be provided to other tools requesting model data on the user's behalf without providing the actual login information.
Standard Workflow	<p>Creating an access token</p> <ul style="list-style-type: none"> • The <i>Model User</i> logs in into the <i>Model Store Server</i> via its <i>Model Store Client</i> and a session is opened. • The <i>Model User</i> calls a synchronous function "create access token" on his <i>Model Store Client</i>, which generates an access token for his account on the <i>Model Store Server</i>. • The <i>Model User</i> is provided the access token, which he can provide to various tools • Any tool configured with the access token can access the same model data as the user until the access token is removed • The <i>Model User</i> logs out of the <i>Model Store Server</i>. <p>Removing an access token</p> <ul style="list-style-type: none"> • The <i>Model User</i> logs in into the <i>Model Store Server</i> via its <i>Model Store Client</i> and a session is opened. • The <i>Model User</i> calls a synchronous function "remove access token" on his client providing the access token information to be removed. • The <i>Model Store Server</i> informs the <i>Model User</i> that the access token has been removed. • Any tool configured with the access token can no longer access the model data on the user's behalf. • The <i>Model User</i> logs out of the <i>Model Store Server</i>.
Comments	Since the <i>Model Store Server</i> should be able to be configured for authentication backends, the login information of the <i>Model User</i> is most likely to be used in several systems. Directly providing this login information to the tools that act on the user's behalf would introduce a security risk.
Resulting Requirements	<i>Model Store Access Token</i>



	<p>ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

2.2.8 Uploading of Models to a Model Store

Name	Uploading of Models to a Model Store
Main Perspective	<i>Model Vendor</i>
Goals	A model is uploaded by the <i>Model Vendor</i> to the <i>Model Store Server</i> in a transactional manner and is immediately available for all <i>Model Users</i> .
Standard Workflow	<ul style="list-style-type: none"> • The <i>Model Vendor</i> creates the model files and the model description file containing all dependencies from other models and puts them into right location of his local model storage. • The <i>Model Vendor</i> logs in into the <i>Model Store Server</i> via its <i>Model Store Client</i> and a session is opened. • The <i>Model Vendor</i> calls a synchronous function "upload package" on its <i>Model Store Client</i>, which requests the location of the file system position of the package. • The <i>Model Store Client</i> transfers the package to the <i>Model Store Server</i>. • The <i>Model Store Server</i> saves the package in its storage component. • The <i>Model Store Server</i> sends a commit message to the <i>Model Store Client</i>. • The <i>Model Store Client</i> informs the <i>Model Vendor</i> about the success of the action.
Comments	<p>The transfer is skipped after some seconds and reported as "transaction fails" in cases the <i>Model Store Server</i> could not be reached or the transaction could not be finished within that time.</p> <p>If a model X already exists (same vendor, model id and version) then it may not be overwritten, which is reported as "error: model X in version ... already exists".</p>
Resulting Requirements	<i>Store components data</i>

2.2.9 Financial processes



Name	Financial processes
Main Perspective	<i>Model Vendor, Model Store Operator</i>
Goals	The <i>Model Vendor</i> is paid for the download of models by <i>Model Users</i> within a pre-defined time frame (e.g. monthly or quarterly bills). The payment is done by the <i>Model Store Operator</i> while a part of the total income is retained for the operation of the <i>Model Store Server</i> .

	<p>ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

Standard Workflow	<p>The <i>Model Store Server</i> runs a UNIX cron like utility. Payment jobs are configured for each of the <i>Model Vendors</i>, when they are registered at the <i>Model Store Server</i>.</p> <p>We assume a monthly payment for a <i>Model Vendor X</i> for the following description of the payment process.</p> <ul style="list-style-type: none"> • The <i>Model Store Server</i> checks regularly for triggers of payment jobs. If a job has to be performed, then it is done according to the following descriptions: • The <i>Model Store Server</i> creates a list of models of <i>Model Vendor X</i>, which have been downloaded since the last billing. The list also contains information identifying <i>Model Users</i>. The base for billing is the list of log-entries created after model download transactions (see also "Payment Scenario"). The bill is finally formatted and transferred to the <i>Model Vendor</i> by e-mail in PDF or/and spreadsheet format. • The <i>Model Store Server</i> initiates a respective transfer of money from the <i>Payment Service</i> to a bank account of the <i>Model Vendor</i>. • The <i>Model Store Server</i> creates a daily report of billings and sends it to the <i>Model Store Operator</i>.
Comments	<p>The <i>Model Store Server</i> and respectively the Administration Client (see Admin CLI in Figure 3 and section 3.2.9) will provide an interface to query passed and future billings.</p> <p>This is an optional use case.</p>
Resulting Requirements	<i>Multiple Model Store Servers in Model Store Client</i>

2.2.10 Account and Access Management

Name	Account and Access Management
Main Perspective	<i>Model Store Operator</i>
Goals	<p>Access to repositories on the Model Store Server can be restricted to specific users and groups of users. For each user or group of users the access can be restricted in terms of read, write and/or update/delete access. A user without permissions for the specific task cannot perform this operation.</p> <p>Configuration and integration of standard authentication backends should be considered.</p>
Standard Workflow	<ul style="list-style-type: none"> • <i>Model Store Operator</i> logs into <i>Model Store Server</i> • <i>Model Store Operator</i> is provided the current configuration • <i>Model Store Operator</i> can create, edit or delete groups

	<p>ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

	<ul style="list-style-type: none"> • <i>Model Store Operator</i> can edit permissions for <ul style="list-style-type: none"> ○ Read permissions for users ○ Write permissions for users ○ Update/Delete permissions for users ○ Read permissions for groups ○ Write permissions for groups ○ Update/Delete permissions for groups
Comments	-
Resulting Requirements	<p><i>Model Store user access management</i></p> <p><i>Model Store access management per repository</i></p>



2.2.11 Model Storage on Server Side

Name	Model Storage on Server Side
Main Perspective	<i>Model Store Server, Model Store Operator</i>
Goals	The <i>Model Store Server</i> is able to store model data locally in a repository. An arbitrary number of repositories can be configured on the server.
Standard Workflow	<ul style="list-style-type: none"> • The <i>Model Store Operator</i> logs in into the <i>Model Store Server</i>. • The <i>Model Store Operator</i> is provided with the current repository information. • The <i>Model Store Operator</i> can add a repository. • The <i>Model Store Operator</i> can delete a repository.
Comments	-
Resulting Requirements	<p><i>Store components data</i></p> <p><i>Store component metadata</i></p>

2.3 Requirements resulting from the Use Cases

The requirements in this section directly result from the use cases in the previous section. They are prioritized by the keywords MUST or SHOULD, meaning that they are either necessary for the function of the Model Store, or that they add functionality, but are not crucial for the basic operation of the Model Store.

No.	Requirement Title and its Description
1	<p><i>Store components data</i></p> <p>The <i>Model Store Server</i> and <i>Model Store Client</i> MUST be able to store components, comprising arbitrary payload data and metadata for the components (e.g. vendor, version, etc).</p>
2	<p><i>Vendors must be able to perform management operations on their packages</i></p> <p>The <i>Model Store Server</i> MUST be able to support the following operations for the vendor stakeholder group: Creation and Delivery of Packages.</p> <p>The <i>Model Store Client</i> MUST be able to support the following operations for the vendor stakeholder group: Delete Packages from the Client Storage, Retrieve and Update of Packages from the Server.</p>
3	<p><i>Download models</i></p> <p>The <i>Model Store Server</i> and <i>Model Store Client</i> MUST be able to support the following operations for the user stakeholder group: Retrieve of models.</p>
4	<p><i>Model Store user access management</i></p> <p>The <i>Model Store Server</i> and <i>Model Store Client</i> MUST be able to incorporate an access management right system supporting users.</p>
5	<p><i>Model Store group access management</i></p> <p>The <i>Model Store Server</i> SHOULD be able to incorporate an access management right system supporting groups.</p>
6	<p><i>Model Store access management per repository</i></p> <p>The <i>Model Store Server</i> MUST be able to restrict the access to components to certain users.</p>
7	<p><i>Store component metadata</i></p> <p>The <i>Model Store Server</i> MUST be able to provide metadata for a requested component.</p>
8	<p><i>Searching for components</i></p> <p>The <i>Model Store Client</i> MUST be able to provide a search functionality for components.</p>

	<p style="text-align: center;">ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

<i>No.</i>	<i>Requirement Title and its Description</i>
9	<p><i>Model Store Payment</i></p> <p>The <i>Model Store Server</i> SHOULD be able to support payment</p>
10	<p><i>Multiple Model Store Servers in Model Store Client</i></p> <p>The <i>Model Store Client</i> MUST be able to be configured for multiple <i>Model Store Servers</i> including prioritization of the <i>Model Store Servers</i> to be queried. It SHOULD be possible to add, changed and deleted this <i>Model Store Servers</i> in the config file from the <i>Model Store Client</i>.</p>
11	<p><i>Model Store Access Token</i></p> <p>The <i>Model Store Server</i> MUST be able to generate access tokens for users.</p>

3 System Design

3.1 Initial Design Decisions

Figure 1 introduced a component description model as one of the possible models stored in a package. There are plenty of possibilities on how this model should be encoded, from STEP Express over XML up to AutomationML. The decision of the project partners was to use AutomationML here, since the partners have experiences and tools to create AutomationML files. Additionally AutomationML has been used in WP2 to describe requirements for production systems and will be used in further WPs e.g. to describe interconnection of simulation models for co-simulation for virtual commissioning of those production systems. AutomationML is publicly available [Am12016a] and standardized by the IEC [IEC62714:2018] as well as its basic format CAEX [IEC6242:2008].

The use of AutomationML here implies informally to use COLLADA for geometry models, since this is recommended by the AutomationML standards suite. FMI-based behavior models should be used, while the referencing of FMUs (Functional Mock-up Units) out of AutomationML has been introduced by another research project [Suess2016].

3.2 Decomposition of Model Store Server and Client

Figure 3 provides an overview about the system architecture of the Model Store. It consists of two main components (ModelStore CLI Client, ModelStore Server), which interact according to the client-server communication pattern.

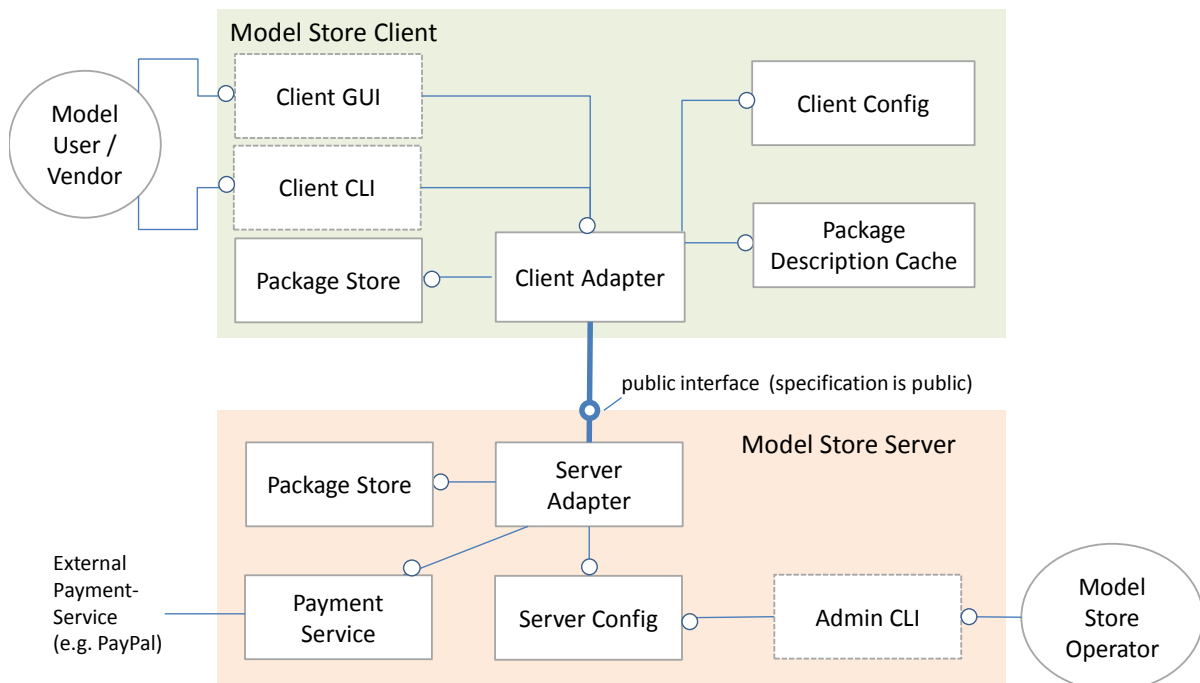


Figure 3: System architecture of the Model Store

Each of the main components consists of inner software modules for better adaptability of the software to new use cases. For example a text console based client application can be easily adapted to become a graphical user interface, if the logic for application configuration and dependency management are implemented as separate software modules. The same approach is used for

implementation of the server application. The following sections describe the responsibility and functionality of the software modules.

In order to be able to implement Model Store Clients and/or Servers independently from the ENTOC project consortium it will be necessary to provide or consume the interface of the Model Store Server (denoted as public interface in Figure 3 and described as ModelStore API in section 3.2.7). Thus it becomes possible e.g. to extend new or existing engineering, planning or simulation tools to access a Model Store.

3.2.1 Client CLI

The Client CLI is a command line based tool to manage models via accessing one or more *Model Store Servers*. For each of the servers the client has to manage access information (user accounts) and the related server address information. The models on the servers have their own life cycle thus the Client CLI has to provide functions to update information about which models are available at the servers and which model versions are supported. Finally it should be possible to download packages to the client side, while the dependencies between packages are calculated on client side.

The server management should be done by manual editing a configuration file or by using the `add_/remove_server` functions of this interface. It may have the following structure:

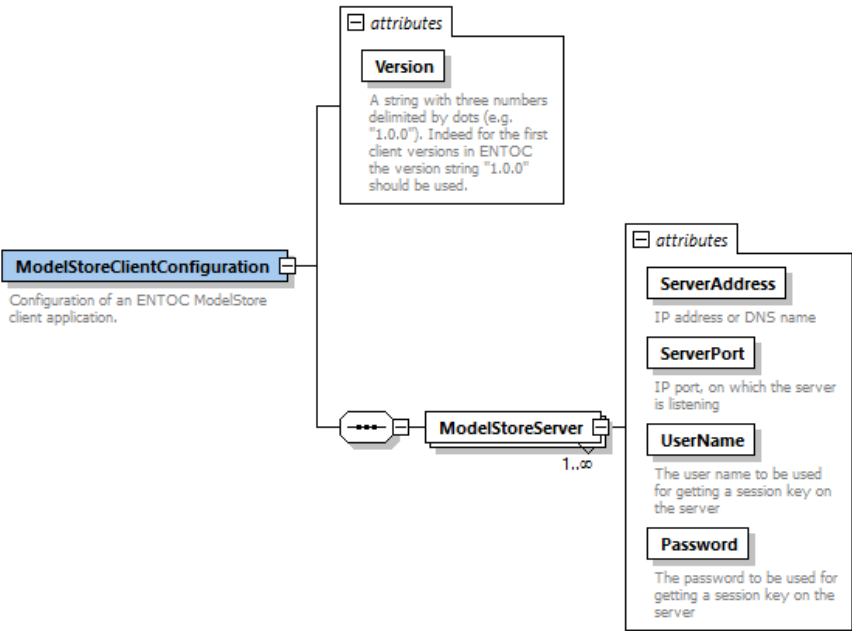




Figure 4: Structure of a configuration file for an ENTOC ModelStore Client

	<p>ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

The Client CLI shall provide functions to be used the user as mentioned in the following, while the syntax is in scope of the CLI implementation. The functions are summarized in the **ClientCommandLineInterface**:

- `get_package(pkg, version)`: This function accesses the *Model Store Server* in order to get all the requested package and all packages it depends on.
- `get_package_dependencies(pkg, version)`: This function accesses the Package Description Cache in order to query, which packages have to be installed in order to fulfil all of the dependencies. An advanced implementation can contact the *Model Store Server* in order to determine the cost information, based on the information, which of the necessary packages have to be acquired newly.
- `update`: It is assumed, that *Model Store Servers* provide meta-information files about the packages they manage. All meta-information files of all the configured *Model Store Servers* are downloaded by using the command "update".
- `upgrade`: The newest available version on any of the configured servers is determined for each package existing on the client side. Those newest versions are then downloaded to the client side including all dependent packages.
- `upgrade(pkg)`: This command upgrades only a single package *pkg* to the newest version.
- `list()`: This command lists all available packages. Per default from the server. Optionally use local as input, to list all model packages in the local storage.
- `vendorlist`: This command lists all available vendors.
- `search(pkg_str)`: This command lists all available package names including the character string *pkg_str*. All configured servers are taken into account.
- `info(pkg)`: This command lists the meta-information of a single package referenced by name given by *pkg*.
- `install(pkg)`: This command installs a package referenced by its name *pkg*.
- `remove(pkg)`: This command removes a package *pkg* on the client side.
- `add_server(url, port, user_name, password)`: this function updates the configuration file by adding a server to the configured list *Model Store Servers*.
- `remove_server(server_url, port)`: this function removes all related entries from the configuration file.
- `change_local_storage(path)`: This command change the location of the storage on the local machine. By default it is "C:\DownloadedPackages", this will be created with the first start of the model store CLI client.
- `--help`: This command lists all available ModelStore commands and a short description of them.
- `login`: This command will direct you to input your username and password. If they are correct you will be able to carry out all other functions mentioned above on the server.

The following functions of the **ClientCommandLineInterface** are available for package administrators only. Package administrators are provided with special access rights on the server side.

- `install(pkg)`: This command installs a new package on the server. The package *pkg* refers to a local package folder on the client side.
- `deprecate(pkg, version)`: This command deprecates a version of a package referenced by the package name *pkg* and the *version*. All lower versions of the package are deprecated too.

Currently there are no functions to uninstall a package from the server. This is only in scope of the *Model Store Server* provider.

The command line tool should be called `cli` or `cli.exe`, so that commands can be written like: `cli upgrade c:\\mypackage.pkg`

3.2.2 Client GUI

The Client GUI is an alternative for the Client CLI. It provides the same functionality as the text-based Client CLI (CCLIUserInterface), but provides a form based interface, which could look like the interface shown in Figure 5.

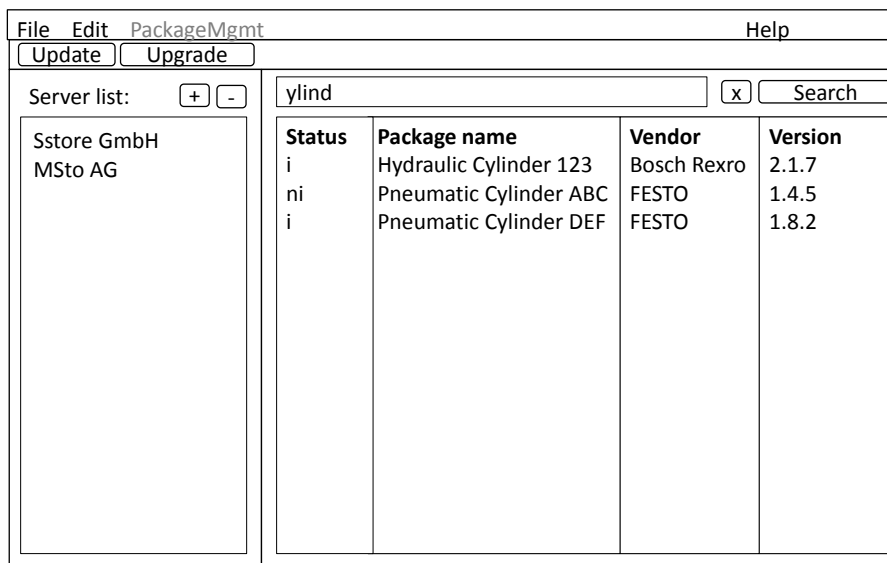




Figure 5: Draft design of the Client GUI

The Client GUI provides direct shorthand's (buttons, lists, tables and text entry fields) for the commands `update`, `upgrade`, `list`, `search` and it provides context menus for `upgrade(pkg)`, `info(pkg)`, `install(pkg)`, `remove(pkg)`. The search result results in a table providing information like install status (here `i`=installed, `ni`=not installed), package name, vendor and version. Package management is available only for package administrators and provides the functionality of `install(pkg)` and `deprecate(pkg)` by providing either file selection dialogs or search-dialogs as basic user interface. Additionally *Model Store Servers* may be added via the GUI.

	<p style="text-align: center;">ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

3.2.3 Client Config

The Client Config module is responsible to access the configuration file as depicted in Figure 4. The purpose of this module is to provide an interface to access *Model Store Server* information directly via an API. Thus the module is responsible to abstract from the structure, encoding and location of the configuration.

The Client Config provides an interface to be used by the CCLI called **ClientConfigurationAdminInterface** with the following functions:

- `add_server(url, port, user_name, password)`: this function updates the configuration file by adding a server to the configured list *Model Store Servers*.
- `remove_server(server_url, port)`: this function removes all related entries from the configuration file.
- `init()`: initially reads the configuration file and creates an internal representation of that file.
- `get_servers()`: provides a list of server objects, each containing information like *Model Store Server* URL, port, user name and password. An initially empty (null) session ID is provided too.

3.2.4 Package Store

The Package Store is a universal component, which is intended to be used in the Model Store Client and the Model Store Server. It provides access to all packages managed locally. The intended implementation strategy is to use the filesystem for package storage.

The Package Store provides the **PackageStoreAccessInterface**, which provides following basic functions:

- `get_package(pkg, version)`: This function returns a zipped package from the local storage without dependencies. It is intended that it may be unzipped at the root folder of another package management file system.
- `put_packages(pkg_zip)`: This function puts a package files from a zipped package into the local file system.
- `remove_package(pkg, version)`: This function removes a package from the local storage.
- `get_list_of_packages()`: This function returns a list of packages in the local storage.
- `get_package_descriptions()`: This function returns a zipped file containing all package description files available in the local storage.

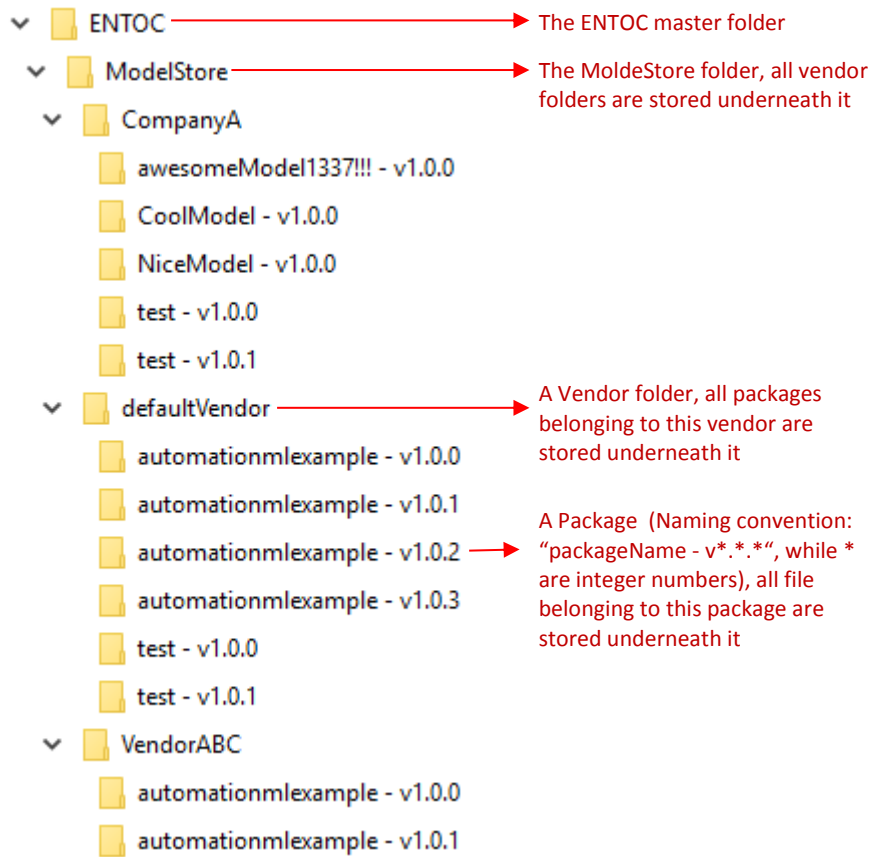




Figure 6: The Folder Structure of the Package Store

3.2.5 Package Description Cache

The Package Description Cache may hold and serve package descriptions of *Model Store Servers*, which can be reached by the Model Store Client. Each package in the package Store needs to include such a package description file called "PackageDescription.json". This file is mandatory for every package. A description of this file is provided by Annex B as a JSON schema. This schema can be used to validate a package description file.

The Package Description Cache provides the **PackageDescriptionCacheInterface** with following functions:

- `put_package_list(server, port, list)`: This function is called when the user triggers the update function and the related *Model Store Servers* provide newer package descriptions than stored locally in the Package Description Cache.
- `get_available_package_descriptions()`: returns a list of `description_objects`, which have attributes like `original_server`, `description_string`, etc.

	<p style="text-align: center;">ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p style="text-align: center;">Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

3.2.6 Client Adapter

This module is the central component of the Model Store Client. It consumes the services of the *ModelStoreServerInterface*, the *ClientConfigurationAdminInterface*, the *PackageStoreInterface* and the *PackageDescriptionCacheInterface*. It implements the business logic available on the client side, including calculation of dependencies between packages. The *ModelStoreServerInterface* is described by using a formal language (swagger) and code for a proxy object may be generated out of this interface description, which can be used to implement parts of the Model Store Client.



The Client Adapter provides a **ClientAdapterUserInterface** which provides all functions of the *ClientCommandLineInterface*, but implemented as API and not as command line based interface to the user.

3.2.7 Server Adapter

The Server Adapter is the central module of the *Model Store Server*. It uses a Package Store as specified in section 3.2.4 for storage of component packages. But it controls access to the packages by using information of a server configuration (Server Config) and a Payment Service.

This module provides an interface called **ModelStore API**, which can be accessed via REST call to the ModelStore Server URI and provides following functions (see Annex A for a formal interface description):

- **InitializeModelStoreServerSession(username, password)**: this function provides a *session_key*, which is used for getting access to all the other functions. (API Call: `.../api/InitializeModelStoreServerSession/username,password`)
- **DisableSessionKey(sessionKey)**: this function disables the session key. (API Call: `.../api/DisableSessionKey/sessionKey`)
- **DownloadPackage(vendorName, packageName, sessionKey)**: This function returns a zipped package from the package store of the *Model Store Server* without dependencies. It is intended that it may be unzipped at the root folder of a Model Store Client. (API Call: `.../api/DownloadPackage/vendorName/packageName/sessionKey`)
- **DownloadPackageInfo(vendorName, packageName, sessionKey)**: This function provides a summary as a JSON object, containing how much data would be transferred for a list of requested packages, how much would it cost and which packages are not available or deprecated. (API Call: `.../api/DownloadPackageInfo/vendorName/packageName/sessionKey`)
- **DownloadPackageDescription(vendorName, packageName, sessionKey)**: This function returns the `packageDescription.json` file belonging to the input package as zipped package. It is also able to return all `PackageDescription.json` for all packages on the *Model Store Server* as zipped package. If you only give the *sessionKey* as input (or the string "all" for *vendorName* and *packageName*). This can be used to fill the *PackageDescriptionCache* of the *Model Store Client* (API Calls: `.../api/DownloadPackageDescription/vendorName/packageName/sessionKey` `.../api/DownloadPackageDescription/sessionKey`)
- **PackageList(vendorName, sessionKey)**: This function returns a JSON object which contains a list of packages belonging to a certain vendor from the package store of the *Model Store Server*. If one want to get of all model package in the package store, one can use "all" as value for *vendorName* and gets back a JSON object containing all model packages clustered by their vendor. (API Call: `.../api/PackageList/vendorName/sessionKey`)
- **VendorList(sessionKey)**: This function returns a list of all vendors available in the package store of the *Model Store Server*. (API Call: `.../api/VendorList/sessionKey`)

	<p style="text-align: center;"> ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015 Project Coordinator: Thomas Bär, Daimler AG </p>	
---	---	---

- `UploadPackage(requestBody, sessionKey)`: This function puts package files from a zipped package into the package store of the *Model Store Server*. It gets its mainly needed data from the `requestBody`. The `requestBody` has to be in JSON format and must include the attributes `packageName`, `vendorName` and `inputFileData`. Otherwise this function will crash. The attribute have to include to following data:
`packageName`, the name the model package should have in the package store of the server as string
`vendorName`, the name of the vendor under which the model package should be stored have in the package store of the server As string
`inputFileData`, the zipped package that includes all the files that are wished to be stored inside the model package
 (API Call: `.../api/UploadPackage/sessionKey`)

3.2.8 Server Config

The Server Config module manages the local storage of the server configuration information as depicted in Figure 7 and Figure 8.

This module provides two interfaces. The first interface is called the **ServerAdminInterface**, which provides the functions as defined for the Admin CLI, but this interface is implemented as API and not as a command line interface.

The second interface is called the **ServerConfigurationRunTimeInterface**, which provides the following functions:

- `start_session(company_id, user_id, password)`: this function returns a `session_id`, which is persistently maintained together with the LastLogin time.
- `stop_session(session_id)`: which maintains the removal of the given `session_id` from a user entry and the LastLogout timestamp.
- `has_role(session_id, role)`: return true/false whether the user identifiable via the given `session_id` has the requested role.

The `ServerAdminInterface` is used by the Admin CLI and the `ServerConfigurationRunTimeInterface` is used by the Server Adapter according to Figure 3.

It is assumed, that the server configuration is stored in a file with the following structure, which is self-explanatory due to the in-line comments:

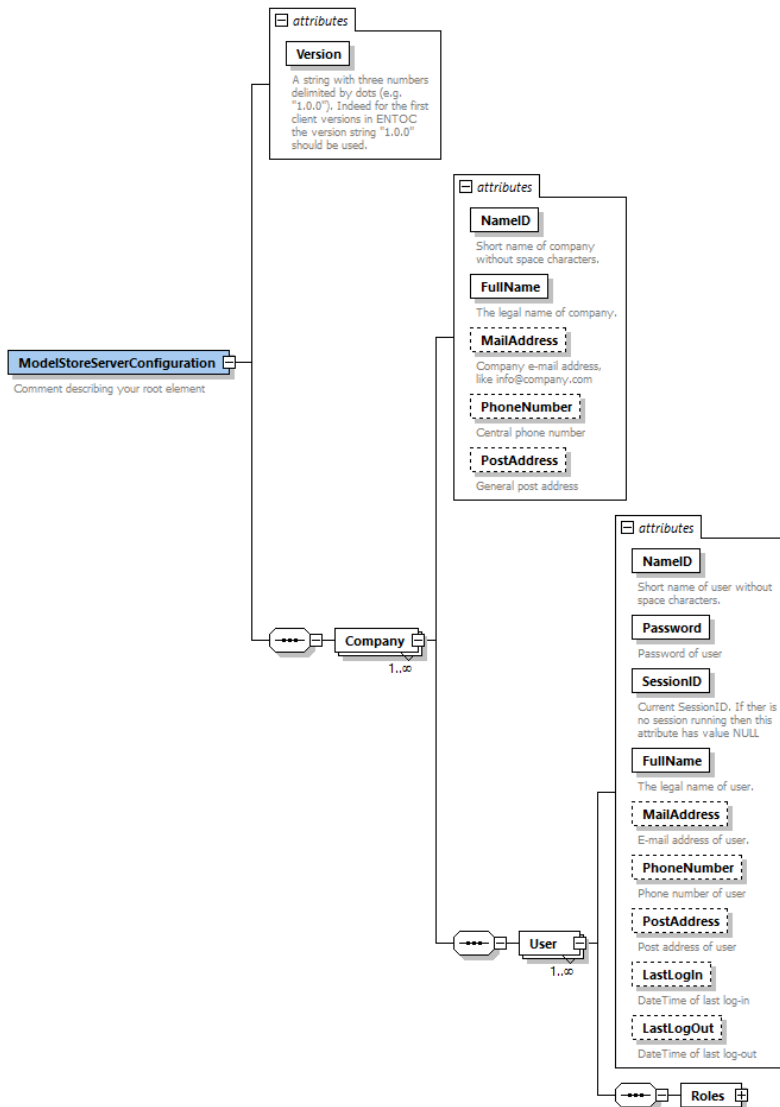


Figure 7: General structure of the Model Store Server configuration file.

Figure 8 provides an overview about the specified user rights. Examples of allowed functions are given in the comments of the user roles.

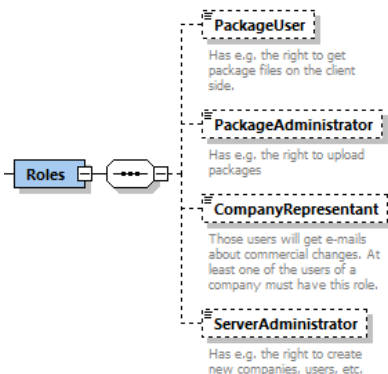




Figure 8: User roles within the Model Store Server configuration file

	<p style="text-align: center;">ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

3.2.9 Admin CLI

The Admin CLI of the Model Store Server provides access to the Server Config module via a command line interface called **ServerAdminCommandLineInterface**. The operator of the Model Store will use this interface to manage clients, users and access rights.

The **ServerAdminCommandLineInterface** provides following functions for the management of companies, users and sessions:

- `add_company(name_id, full_name, mail_address, phone_number, post_address)`
- `update_company(name, mail_address, phone_number, post_address)`
- `remove_company(name_id)`
- `add_user(company_id, name_id, password, full_name, mail_address, phone_number, post_address, roles)`
- `update_user(company_id, name_id, password, full_name, mail_address, phone_number, post_address, roles)`
- `remove_user(company_id, name_id)`
- `remove_package(pkg, version)`: This function removes a package with the name *pkg* and the related *version* from the package store of the *Model Store Server*.

The mentioned functions are self-explanatory by the provided function and argument names. They add/update/remove entries in the server configuration file.

3.2.10 Payment Service

The payment service is responsible to manage money transfer for delivered packages. It provides the `PaymentInterface` with the following functions:

- `transfer_money(sending_company, receiving_company, cost, order_id)`: This is a complex function, which logs, which packages have been ordered by which user. The order gets an `order_id` and the data set is stored in a payment service internal database (which could be file-based too). The concrete payment is delegated to on-line payment services like PayPal. Thus the arguments `sending_company` and `receiving_company` represent all the necessary address information needed from the on-line payment service. Those data have to be managed in the server configuration file.

A structure of a database to store commercial model transfer data is self-explanatory depicted in Figure 9.

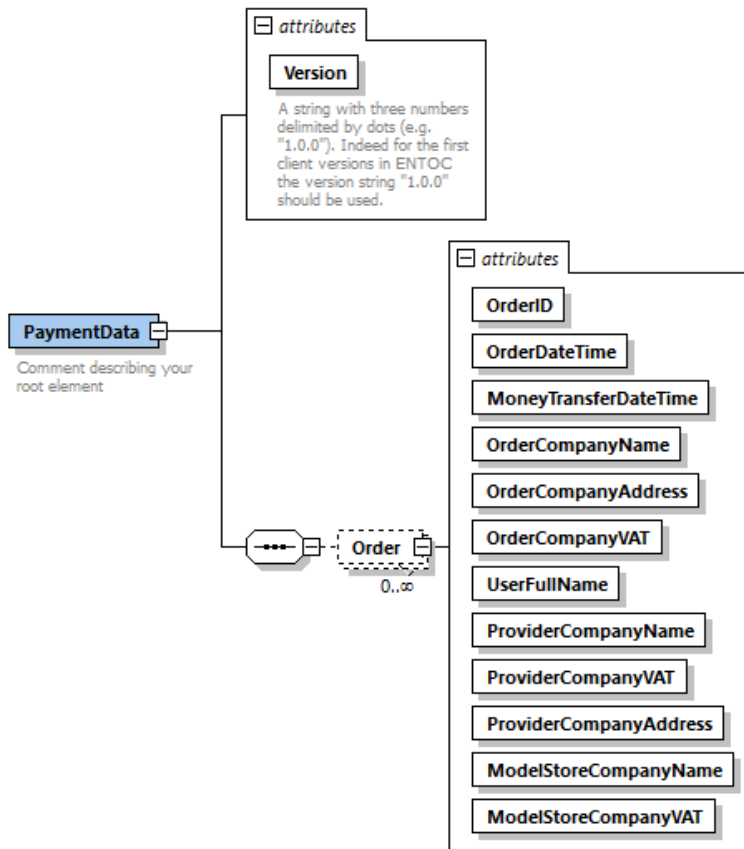






Figure 9: Example for a payment data store

	<p style="text-align: center;">ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

4 Summary and Outlook

This document provides a specification of a Model Store, which is necessary for exchange of engineering models between different companies. Those models are related to mechatronic components, which are assembled to production systems. The concept of packages has been introduced, while a package can contain multiple models as files (e.g. geometry models and behavior models), which are described within a package description file. The Model Store is designed as client-server application. The client specification requires the management of package dependencies if e.g. other component models are aggregated to a system model.

A reference implementation of the model store is one of the next tasks within the ENTOC project (Task 3.3). It will be evaluated within application driven use cases in WP6 of the ENTOC project.

	<p style="text-align: center;"> ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015 Project Coordinator: Thomas Bär, Daimler AG </p>	
---	---	---

References

- [Aml2016a] AutomationML e.V.: Whitepaper AutomationML, Part 1 – Architectural and general requirements, Version 2.0.0, April 2016.
- [D3.1] ENTOC consortium: Deliverable D3.1: Package Format specification. May 2018.
- [IEC62714:2018] IEC: Engineering data exchange format for use in industrial automation systems engineering - Automation Markup Language - Part 1: Architecture and general requirements
- [IEC6242:2008] IEC: Representation of process control engineering – Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools. IEC 6242/Ed.1.
- [Suess2016] Süß, S.; Magnus, St.; Thron, M.; Zipper, H.; Odefey, U.; Strahilov, A.; Klodowski, A.; Bär, Th.: Test methodology for virtual commissioning based on behaviour simulation of production systems. 21st IEEE International Conference on Emerging Technologies and Factory Automation ETFA 2016, Berlin, September 2016.

Annex A: Formal Description of the Model Store Server Interface

The following text describes the interface of the ServerAdapter formally as a swagger file (<https://swagger.io/>), which makes it easy to generate a REST interface implementation:

```
{
  "swagger": "2.0",
  "info": {
    "version": "v1",
    "title": "ModelStore API",
    "description": "This the description of the Model Store Server API.",
    "contact": {
      "name": "ifak - Institut für Automation und Kommunikation e. V.",
      "url": "https://www.ifak.eu/",
      "email": "info@ifak.eu"
    }
  },
  "paths": {
    "/api/DisableSessionKey/{sessionKey}": {
      "put": {
        "tags": [
          "DisableSessionKey"
        ],
        "operationId": "ApiDisableSessionKeyBySessionKeyPut",
        "consumes": [],
        "produces": [
          "text/plain",
          "application/json",
          "text/json"
        ],
        "parameters": [
          {
            "name": "sessionKey",
            "in": "path",
            "required": true,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "Success",
            "schema": {
              "type": "string"
            }
          }
        }
      }
    },
    "/api/DownloadPackage/{vendorName}/{packageName}/{sessionKey}": {
      "get": {
        "tags": [
          "DownloadPackage"
        ],
        "operationId":
"ApiDownloadPackageByVendorNameByPackageNameBySessionKeyGet",
        "consumes": [],
        "produces": [
          "application/zip"
        ],
        "parameters": [
          {
            "name": "vendorName",
            "in": "path",
```



ENTOC
Engineering tool chain for efficient and iterative
development of smart factories
ITEA 3, 15015



Project Coordinator: Thomas Bär, Daimler AG

```
        "required": true,
        "type": "string"
    },
    {
        "name": "packageName",
        "in": "path",
        "required": true,
        "type": "string"
    },
    {
        "name": "sessionKey",
        "in": "path",
        "required": true,
        "type": "string"
    }
],
"responses": {
    "200": {
        "description": "Success",
        "schema": {
            "type": "string"
        }
    }
}
},
"/api/DownloadPackageDescription/{vendorName}/{packageName}/{sessionKey}":
{
    "get": {
        "tags": [
            "DownloadPackageDescription"
        ],
        "operationId":
"ApiDownloadPackageDescriptionByVendorNameByPackageNameBySessionKeyGet",
        "consumes": [],
        "produces": [
            "application/zip"
        ],
        "parameters": [
            {
                "name": "vendorName",
                "in": "path",
                "required": true,
                "type": "string"
            },
            {
                "name": "packageName",
                "in": "path",
                "required": true,
                "type": "string"
            },
            {
                "name": "sessionKey",
                "in": "path",
                "required": true,
                "type": "string"
            }
        ],
        "responses": {
            "200": {
                "description": "Success",
                "schema": {
                    "type": "object"
                }
            }
        }
    }
}
```



```
    }
  }
}
},
"/api/DownloadPackageDescription/{sessionKey}": {
  "get": {
    "tags": [
      "DownloadPackageDescription"
    ],
    "operationId": "ApiDownloadPackageDescriptionBySessionKeyGet",
    "consumes": [],
    "produces": [
      "application/zip"
    ],
    "parameters": [
      {
        "name": "sessionKey",
        "in": "path",
        "required": true,
        "type": "string"
      }
    ],
    "responses": {
      "200": {
        "description": "Success",
        "schema": {
          "type": "object"
        }
      }
    }
  }
},
"/api/DownloadPackageInfo/{vendorName}/{packageName}/{sessionKey}": {
  "get": {
    "tags": [
      "DownloadPackageInfo"
    ],
    "operationId":
"ApiDownloadPackageInfoByVendorNameByPackageNameBySessionKeyGet",
    "consumes": [],
    "produces": [
      "text/plain",
      "application/json",
      "text/json"
    ],
    "parameters": [
      {
        "name": "vendorName",
        "in": "path",
        "required": true,
        "type": "string"
      },
      {
        "name": "packageName",
        "in": "path",
        "required": true,
        "type": "string"
      },
      {
        "name": "sessionKey",
        "in": "path",
        "required": true,
```





```
        "type": "string"
      }
    ],
    "responses": {
      "200": {
        "description": "Success",
        "schema": {
          "type": "object"
        }
      }
    }
  }
},
"/api/InitializeModelStoreServerSession/{userName}/{password}": {
  "get": {
    "tags": [
      "InitializeModelStoreServerSession"
    ],
    "operationId":
"ApiInitializeModelStoreServerSessionByUserNameByPasswordGet",
    "consumes": [],
    "produces": [
      "text/plain",
      "application/json",
      "text/json"
    ],
    "parameters": [
      {
        "name": "userName",
        "in": "path",
        "required": true,
        "type": "string"
      },
      {
        "name": "password",
        "in": "path",
        "required": true,
        "type": "string"
      }
    ],
    "responses": {
      "200": {
        "description": "Success",
        "schema": {
          "type": "string"
        }
      }
    }
  }
},
"/api/PackageList/{vendorName}/{sessionKey}": {
  "get": {
    "tags": [
      "PackageList"
    ],
    "summary": "Returns a JSON file that holds a list of all P´packages
provided by a vendor.",
    "description": "Either give a certain vendor as input or just use
\"all\" to get back a JSON with all packages clustered by vendor.",
    "operationId": "ApiPackageListByVendorNameBySessionKeyGet",
    "consumes": [],
    "produces": [
      "text/plain",
```



```
        "application/json",
        "text/json"
    ],
    "parameters": [
        {
            "name": "vendorName",
            "in": "path",
            "required": true,
            "type": "string"
        },
        {
            "name": "sessionKey",
            "in": "path",
            "required": true,
            "type": "string"
        }
    ],
    "responses": {
        "200": {
            "description": "Success",
            "schema": {
                "type": "object"
            }
        }
    }
},
"/api/UploadPackage/{sessionKey}": {
    "post": {
        "tags": [
            "UploadPackage"
        ],
        "operationId": "ApiUploadPackageBySessionKeyPost",
        "consumes": [
            "application/json-patch+json",
            "application/json",
            "text/json",
            "application/*+json"
        ],
        "produces": [
            "text/plain",
            "application/json",
            "text/json"
        ],
        "parameters": [
            {
                "name": "Parameters",
                "in": "body",
                "required": false,
                "schema": {
                    "type": "object"
                }
            },
            {
                "name": "sessionKey",
                "in": "path",
                "required": true,
                "type": "string"
            }
        ],
        "responses": {
            "200": {
                "description": "Success",
```



```
        "schema": {
          "type": "string"
        }
      }
    }
  },
  "/api/VendorList/{sessionKey}": {
    "get": {
      "tags": [
        "VendorList"
      ],
      "operationId": "ApiVendorListBySessionKeyGet",
      "consumes": [],
      "produces": [
        "text/plain",
        "application/json",
        "text/json"
      ],
      "parameters": [
        {
          "name": "sessionKey",
          "in": "path",
          "required": true,
          "type": "string"
        }
      ],
      "responses": {
        "200": {
          "description": "Success",
          "schema": {
            "type": "array",
            "items": {
              "type": "string"
            }
          }
        }
      }
    }
  }
},
"definitions": {}
}
```

	<p>ENTOC Engineering tool chain for efficient and iterative development of smart factories ITEA 3, 15015</p> <p>Project Coordinator: Thomas Bär, Daimler AG</p>	
---	---	---

Annex B: Package Description File Format

The following text describes the JSON schema (<http://json-schema.org/>) of the PackageDescription-file, which makes it easy to validate a package description file provided by a model vendor:

```
{
  "title": "modelPackageDescription",
  "type": "object",
  "properties": {
    "descriptorVersion": {
      "type": "string",
      "pattern": "(\\d+\\.)(\\d+\\.)(\\d+)",
      "title": "The Descriptor version Schema ",
      "description": "informs about the version of the package description
file, in case there are addition or attribute name changes etc. in the future",
      "examples": [
        "1.0.0"
      ]
    },
    "name": {
      "type": "string",
      "pattern": "^[A-Z a-z]{1,} - v(\\d+\\.)(\\d+\\.)(\\d+)",
      "title": "The Name Schema ",
      "description": "the name of the model package",
      "examples": [
        "TheNiceModelPackage - v1.0.7"
      ]
    },
    "version": {
      "type": "string",
      "pattern": "(\\d+\\.)(\\d+\\.)(\\d+)",
      "title": "The Version Schema ",
      "description": "the version of the model package",
      "examples": [
        "1.0.7"
      ]
    },
    "deprecated": {
      "type": "boolean",
      "title": "The Deprecated Schema ",
      "description": "declares whether the package is deprecated or not",
      "default": false,
      "examples": [
        false
      ]
    },
    "previousVersion": {
      "type": "string",
      "pattern": "(\\d+\\.)(\\d+\\.)(\\d+)",
      "title": "The Previous version Schema ",
      "description": "the previous version of the model package",
      "examples": [
        "1.0.5"
      ]
    },
    "description": {
      "type": "string",
      "title": "The Description Schema ",
      "description": "a short description of the model package",
      "default": "",
      "examples": [
        "This is a nice short description of the Model Package."
      ]
    }
  }
}
```



```
    ],
    },
    "typeIdentification": {
      "type": "object",
      "description": "this is needed because the modelPackageType itself is
dependent on its identifier and so are the attributes",
      "patternProperties": {
        "^.*$": {
          "anyOf": [
            {
              "type": "object",
              "description": "",
              "additionalProperties": {
                "modelPackageType": {
                  "type": "string",
                  "title": "The Model package type Schema ",
                  "description": "the type of the model package
(relevant for the search filter implementation)",
                  "default": "",
                  "examples": [
                    "rollerConveyor"
                  ]
                },
              },
              "modelPackageTypeId": {
                "type": "string",
                "title": "The Model package typeid Schema ",
                "description": "the id of the model package
type, this is not always necessary",
                "default": "",
                "examples": [
                  "13-47-42-11"
                ]
              },
            },
            {
              "type": "object",
              "description": "here all the attributes are
stored which are used for the search filter, a clear set of mandatory and optional
attributes should by defined per modelPackageType",
              "patternProperties": {
                "^.*$": {
                  "anyOf": [
                    {
                      "type": "string"
                    },
                    {
                      "type": null
                    }
                  ]
                },
              },
              "additionalProperties": false
            }
          ]
        }
      }
    },
    "additionalProperties": false
  },
  "behaviorModels": {
    "type": "object",
    "description": "the behavior model included in the model package",
    "patternProperties": {
```




ENTOC
Engineering tool chain for efficient and iterative
development of smart factories
ITEA 3, 15015



Project Coordinator: Thomas Bär, Daimler AG

```
    "^.*$": {
      "anyOf": [
        {
          "type": "object",
          "description": "properties can be breaks, conflicts,
replaces, deprecates, contributes",
          "patternProperties": {
            "(breaks)|(conflicts)|(replaces)|(deprecates)|(contributes)": {
              "anyOf": [
                {
                  "type": "string",
                  "title": "The dependency relation
Schema",
                  "description": "The string describes
the path to the depending behavior model file. Hint: This structure could be
extended to 3dModels and others in further iterations of this JSON schema. If such
a dependency relation is needed for this files in the future."
                }
              ]
            }
          }
        }
      ]
    },
    "3dModels": {
      "type": "array",
      "items": {
        "type": "string",
        "title": "The 3dModels item Schema ",
        "description": "the 3d models included in a model package",
        "examples": [
          "test.dae"
        ]
      }
    },
    "others": {
      "type": "array",
      "items": {
        "type": "string",
        "title": "The others item Schema ",
        "description": "other files included in a model package",
        "examples": [
          "hello.xlsx",
          "manual.pdf"
        ]
      }
    },
    "dependencies": {
      "type": "array",
      "items": {
        "type": "string",
        "title": "The dependency item Schema ",
        "description": "The paths of all model packages the actual model
package depends on are stored. The path consists of the folder of the vendor and
the model package itself.",
        "examples": [
          "vendorABC/testPackage - v1.0.4"
        ]
      }
    }
  },
```



ENTOC
Engineering tool chain for efficient and iterative
development of smart factories
ITEA 3, 15015



Project Coordinator: Thomas Bär, Daimler AG

```
"author": {
  "type": "string",
  "title": "The Author Schema ",
  "description": "The author of a model package.",
  "default": "no author given",
  "examples": [
    "Terence Hill"
  ]
},
"createdOn": {
  "type": "string",
  "format": "date-time",
  "title": "The Created on Schema ",
  "description": "The timestamp when a model package was uploaded.",
  "default": "",
  "examples": [
    "2018-04-24T11:01:26.9755646+02:00"
  ]
}
},
"additionalProperties": false,
"required": [
  "descriptorVersion",
  "name",
  "version",
  "deprecated",
  "previousVersion",
  "description",
  "typeIdentification",
  "behaviorModels",
  "3dModels",
  "others",
  "dependencies",
  "author",
  "createdOn"
]
}
```