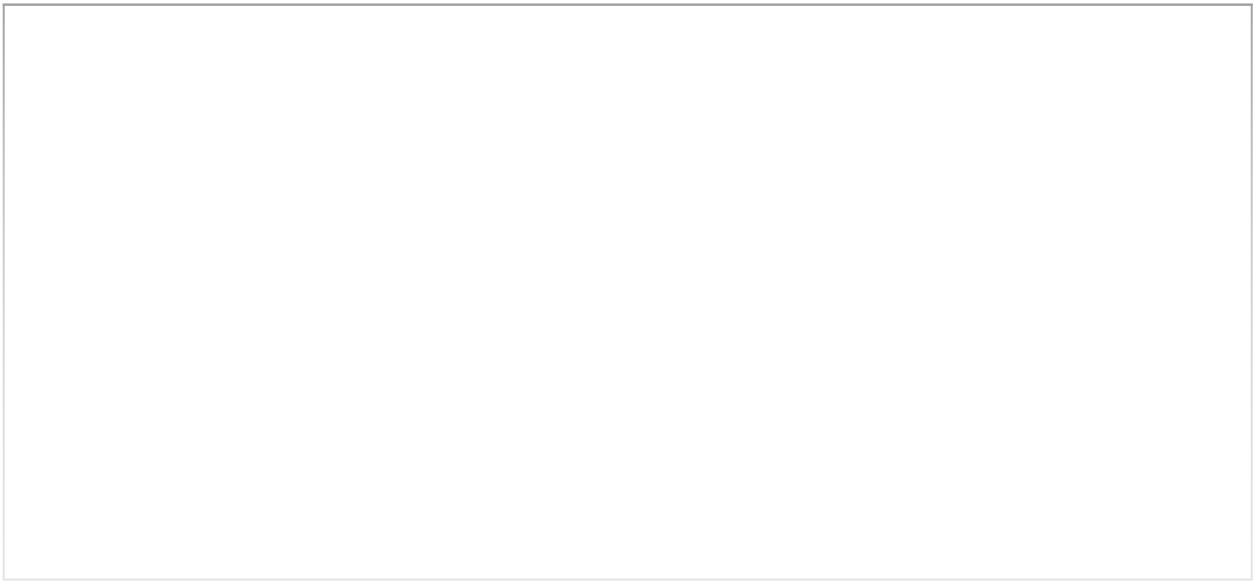
Logical models

Explanatory Notes

There is only a single meta model but there are multiple logical models. Logical models prepare the implementation of certain components of the meta model. The current version of the MedMij Framework has three logical models. Each of them is required for one or more specific implementation components in the MedMij Framework. This relates to the following components:

- the four lists published by *MedMij Registration*: *Data Service Names List*, *OAuthclientlist*, *Whitelist* and *Care Providers List*;
- the *Catalogue* of *Data Services*to be published in the MedMij Framework;
- the (*Hostname* of the) *MedMijSystemNode*to be published in the MedMij Framework.

The four lists have been combined into a single logical model, under the class *MedMijManagementlist*, because they share two characteristics: a timestamp and a serial number.

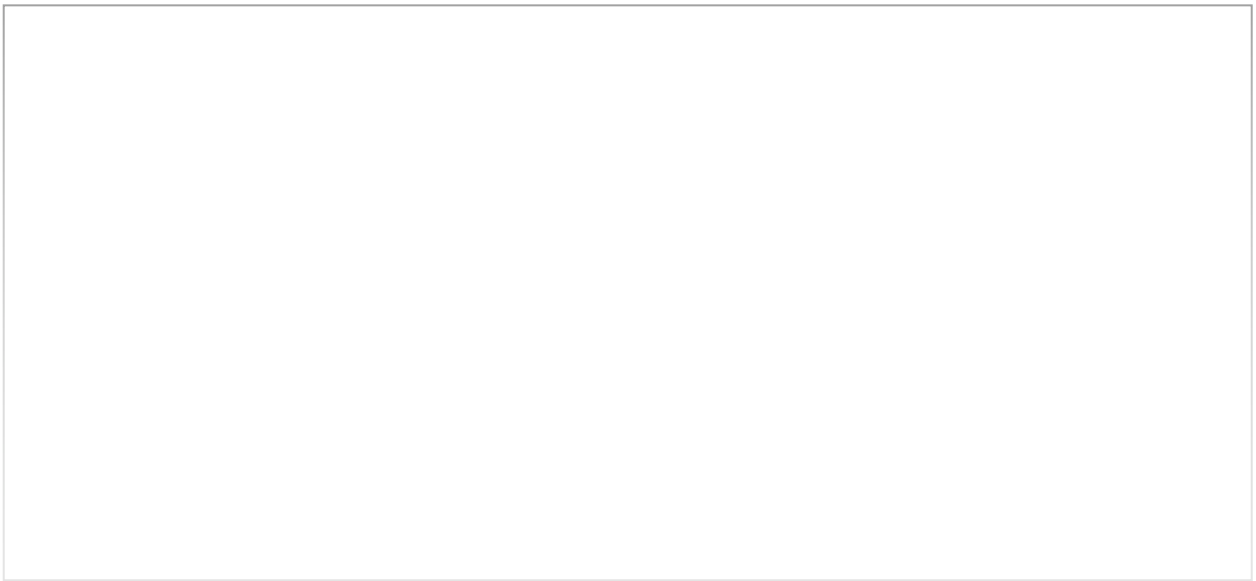


Logical models obey the meta model but specify it. In the step from meta model to logical mode, (logical and other) classes, invariants and basic classes may be added. However, the logical models primarily build on the meta model by using its classes and attributes. In that case, logical classes, values and basic classes accordingly have corresponding classes in the meta model. The similarities are shown below with the logical model named in a table. Where the table for a certain logical class, value or basic class does not name the similarity with the meta model, it means that this is new for the logical level.

Logical classes have fewer or more attributes than the corresponding classes in the meta model. Where there are fewer, this means that the omitted attributes do not need to be included in the component to be implemented, for example of a list to be published. Where there are more, these attributes are passed down from a class in the meta model that the corresponding class in that meta model was existence-dependent on. In the meta model, the last-named class was accordingly accessible for the existence-dependent class but it is no longer present in the specific logical model and thus is no longer accessible either. This means that if the relevant class in the logical model has not taken over the attribute then this will be lost.

Where a invariant from the meta model fits within the scope of the specific logical model, this also appears as an invariant with the logical model, although the formulation will have been adapted to the organisation and naming used in the logical model. In addition, new invariants too may appear on the logical level. Most of them are inheritances: in the step from the meta model to a logical mode, links between classes become broken. If these links are important after all in the logical model then attributes from the meta model are bequeathed from a certain class in the meta model to a lower class, for which a pendant does actually occur in the logical model. Here, "lower class" refers to the fact that this is existence-dependent on the other (higher) class. Such an inheritance invariant is written with a ←. In front of that arrow we see the inheriting attribute of the *logical* class, and behind it we see the path *in the meta model* to the bequeathing class.

Where applicable, the basic classes too from the meta model are taken over by the logical model. There is a single place in the logical model where new basic classes appear as well.



The logical models have a structure that is more oriented towards implementation than the meta model is. This meta model is based on association classes and existence-dependency, whereas the logical models are more hierarchical in nature. Hierarchy is a constriction of associative existence-dependency but is a better fit with many types of standard

implementation technology, this certainly including XML, in which the four lists are implemented. This constriction does however mean that the logical models are less long-lasting and less expandable than the meta model; something that for the meta model is a simple expansion can correspond to a substantial modification of the logical models. This is the price you pay for hierarchy.

When translating the associativity of the meta model into the hierarchy of the logical models, a number of rules of thumb have been applied:

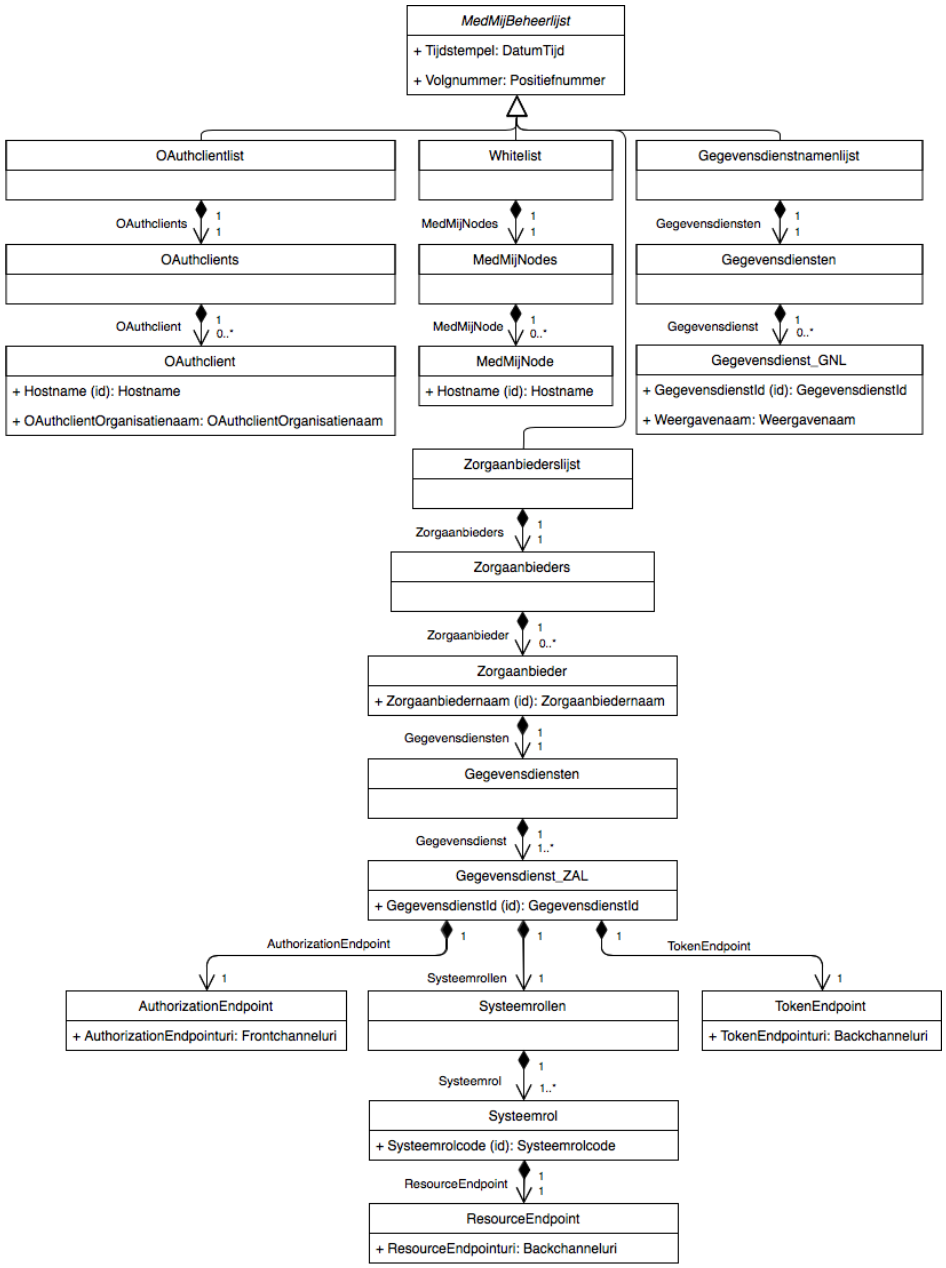
- The top of the hierarchy of a logical model is determined by the scope of the implementation component. The *Care Providers List*, for example, lists first of all the *Care Providers*. Starting from that "logical centre", the hierarchy descends from top to bottom, without exceeding the scope of the implementation. In the logical model, the step towards the bottom in the hierarchy typically takes the form of a 'uses' relationship (the dotted-line arrow).
- En route, a composition hierarchy is built, and in each step a selection is made from the attributes available in the meta model, on the basis of the scope of the implementation component. In doing so, logical classes are not combined into a granular class, not even if no attribute at all is left over. The class granularity of the logical model is accordingly comparable with that of the meta model.
- In addition, as described above, attributes in the meta model that threaten to fall outside its scope but that are actually needed are bequeathed to within the scope. Where this is done, the inheritance is specified in the list of logical invariants.
- Lower classes in the uses hierarchy lie completely within the logical scope of the higher one. In this way, a hierarchy also creates closed "name spaces". This means that their naming is simpler and shorter than in the meta model, where it is precisely the case that all contexts are open-ended. In the logical models, therefore, the names of the classes do not become meaningful until higher classes are considered with them. However, this does simplify the implementation. In a separate table for each logical model, it is ensured that these name changes do not cause the link with the meta model to become lost.
- In a single instance, the previous point has the consequence that there is a risk that a homonym could arise within a single logical model (namely *Data Service* in the lists' logical model). In that case, the names will be expanded so that their hierarchical context becomes visible (namely to *Dataservice_DSNL* and *Dataservice_CPL*).

Note that the uses hierarchy places the existence-dependent relationship upside down. In the corresponding classes in the meta model, in the uses relationship the used class is placed above the using class, whereas the reverse is true in the logical models. This characterises the decisive difference between the meta model's conceptual way of thinking and the logical models' build-oriented way of thinking. When it comes to making the MedMij Framework both consistent and long-lasting, it makes sense to place the meta model centre-stage in respect of model management and then to keep the logical models consistent with it. In this way, the meta model also ensures that the various logical models remain consistent in the long term too. In

fact, the trustworthiness and interoperability that the MedMij Framework has to deliver is dependent on this consistency.

Lists

Logical model



Logical invariants

Relates to instances of logical class ...	Invariant	Component	Explanatory Notes	Nature	Origin
AuthorisationEndpoint	For each *AuthorisationEndpoint a* it is true that: *a . AuthorisationEndpoint uri* ← combination of *a . MedMijNode. ParticipantNode. Node. Hostname* and *a.AuthorisationEndpointpath*, in accordance with the addressing responsibilities on the page Data and performance in UCI Compile and UCI Share	*Care Providers List*	See the page Data and performance in UCI Compile and UCI Share.	inheritance	logical model
Dataservice_CPL	For each *Dataservice_CPL g* with its corresponding *CareproviderDataservice z* it is true that: *g.DataserviceId ← z.Dataservice.DataserviceId*	*Care Providers List*	In this way, the *Care Providers List* inherits the *DataserviceIds* of the *Catalogue*.	inheritance	logical model
Data Service Names List	There is precisely one instance of this.	*Data Service Names List*	This is a loner in the model.	numerical relationship	logical model
MedMijNode	For each *MedMijNode m* it is true that*:* *m.Hostname = m.ParticipantNode .Node.Hostname*	*Whitelist*	In this way, the *MedMijNode* inherits the *Hostname* of the *Node* that it is.	inheritance	logical model
MedMijNode	The hostname of a *MedMijNode* contains a domain name that is a fully-qualified domain name, in accordance with RFC3696, section 2.	*Whitelist*	This is a measure to combat risk 4.4.1.4 from RFC 6819.	local dependency	meta model (with *Node*)

Relates to instances of logical class ...	Invariant	Component	Explanatory Notes	Nature	Origin
OAuthclient	For each *OAuthclient o*: *o.OAuthclientOrganisationname* complies with the OAuthclient names policy.	*Application*	See the OAuthclient names policy.	local dependency	meta model (with *OAuthclient*)
OAuthclient	*For each OAuthclient o it is true that: o.Hostname ← o.MedMijNode .Hostname*.	*OAuthclientlist*	In this way, the *OAuthclientlist* inherits the *Hostnames* of the *Nodes*.	inheritance	logical model
OAuthclientlist	There is precisely one instance of this.	*OAuthclientlist*	This is a loner in the model.	numerical relationship	logical model
ResourceEndpoint	For each *ResourceEndpoint r* it is true that: *r.ResourceEndpointuri* ← combination of *r.MedMijNode.ParticipantNode.Node.Hostname*, *r.ResourceEndpointport and r.ResourceEndpointpath*, in accordance with the addressing responsibilities on the page Data and performance in UCI Compile and UCI Share .	*Care Providers List*	See the page Data and performance in UCI Compile and UCI Share.	inheritance	logical model

Relates to instances of logical class ...	Invariant	Component	Explanatory Notes	Nature	Origin
System Role	For each *System Role s* with its corresponding *CareproviderDataserviceSystemrole z* it is true that: *s.Systemrolecode* ← *z.Systemrole.Systemrolecode*	*Care Providers List*	In this way, the *Care Providers List* inherits the *Systemrolecodes* of the *Register of Information Standards*.	inheritance	logical model
TokenEndpoint	For each *TokenEndpoint t* it is true that: *t.TokenEndpointuri* ← combination of *t.MedMijNode.ParticipantNode.Node.Hostname*, *t.TokenEndpointport* and *t.TokenEndpointpath*, in accordance with the addressing responsibilities on the page Data and performance in UCI Compile and UCI Share.	*Care Providers List*	See the page Data and performance in UCI Compile and UCI Share.	local dependency	logical model
Whitelist	There is precisely one instance of this.	*Whitelist*	This is a loner in the model.	numerical relationship	logical model
Care Providers List	There is precisely one instance of this.	*Care Providers List*	This is a loner in the model.	numerical relationship	logical model

Logical basic classes

Basic class	De
Backchanneluri	See the addressing responsibilities on the page Data and performance in UCI C accordance with RFC3696, section 2.

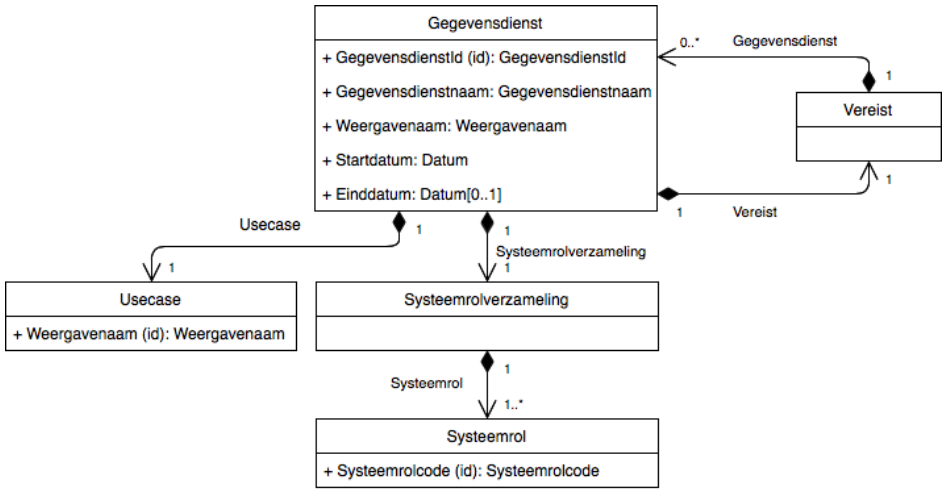
Basic class	De
DateTime	In accordance with the type xs:dateTime , as specified in XML schema 1.0 and inc
Frontchanneluri	See the addressing responsibilities on the page Data and performance in UCI Co accordance with RFC3696, section 2.
DataserviceId	String of at least one, and a maximum of 30, character(s).
Hostname	See the addressing responsibilities on the page Data and performance in UCI Co
OAuthclientOrganisationname	In accordance with applicable Oauthclient names policy.
Positivenumber	A whole number that is not equal to 0.
Systemrolecode	String of at least one, and a maximum of 30, character(s).
Depictionname	String of at least three, and a maximum of 50, character(s).
Careprovidername	In accordance with applicable Care Providersnamespolicy.

Link with meta model

Class in logical model	Origin class in meta model
AuthorisationEndpoint	*AuthorisationEndpoint*
Dataservice_DSNL	*Data Service*
Dataservice_CPL	*CareproviderDataservice*
MedMijNode	*MedMijNode*
OAuthclient	*OAuthclient*
ResourceEndpoint	*ResourceEndpoint*
System Role	*CareproviderDataserviceSystemrole*
TokenEndpoint	*TokenEndpoint*
Careprovider	*Careprovider*

Catalogue

Logical model



Logical invariants

Relates to instances of class ...	Invariant	Component	Explanatory Notes	Nature	Origin
Usecase	For each *Usecase u* it is true that: *u.Depictionname* = "Compile" OR *u.Depictionname* = "Share"	*Catalogue*	This links the names of the subclasses to the depiction names.	local dependency	meta model (with *Usecase*)

Logical basic classes

Basic class	Definition	Origin
Date	In accordance with the type xs:date, as specified in XML schema 1.0.	meta model
DataserviceId	String of at least one, and a maximum of 30, character(s).	meta model
Dataservicename	String of at least three, and a maximum of 50, character(s).	meta model
Systemrolecode	String of at least one, and a maximum of 30, character(s).	meta model

Basic class	Definition	Origin
Depictionname	String of at least three, and a maximum of 50, character(s).	meta model

Link with meta model

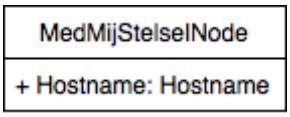
Class/value in logical model	Origin class in meta model
Data Service	*Data Service*
Usecase	*Usecase* , *CompileUsecase* and *ShareUsecase*

Explanatory Notes

The class *Usecase* is an abstract class in the meta model. However, concrete classes are needed in the logical model, namely in the composition hierarchy. In the context of the *Catalogue*, we are not interested here in the whole semantics of the conceptual classes *CompileUsecase* and *ShareUsecase* but only in their respective instances, with the *Depictionname* that they receive from the abstract class *Usecase*, by means of an invariant. In this logical model, this is why we use a concrete class *Usecase* that instantiates to these two.

MedMijSystemNode

Logical model



Logical invariants

Relates to instances of class ...	Invariant	Component	Explanatory Notes	Nature

Relates to instances of class ...	Invariant	Component	Explanatory Notes	Nature
MedMijSystemNode	For the *MedMijSystemNode m* it is true that: *m.Hostname* ← *m.Node.Hostname*	*MedMijSystemNode*	In this way, the *MedMijSystemNode*, inherits - from the *Node* that it is - the *Hostname*.	inheritance

Logical basic classes

Basic class	Definition	Origin
Hostname	See the addressing responsibilities on the page Data and performance in UCI Compile and UCI Share.	meta model

Link with meta model

Class in logical model	Origin class in meta model
MedMijSystemNode	*MedMijSystemNode*