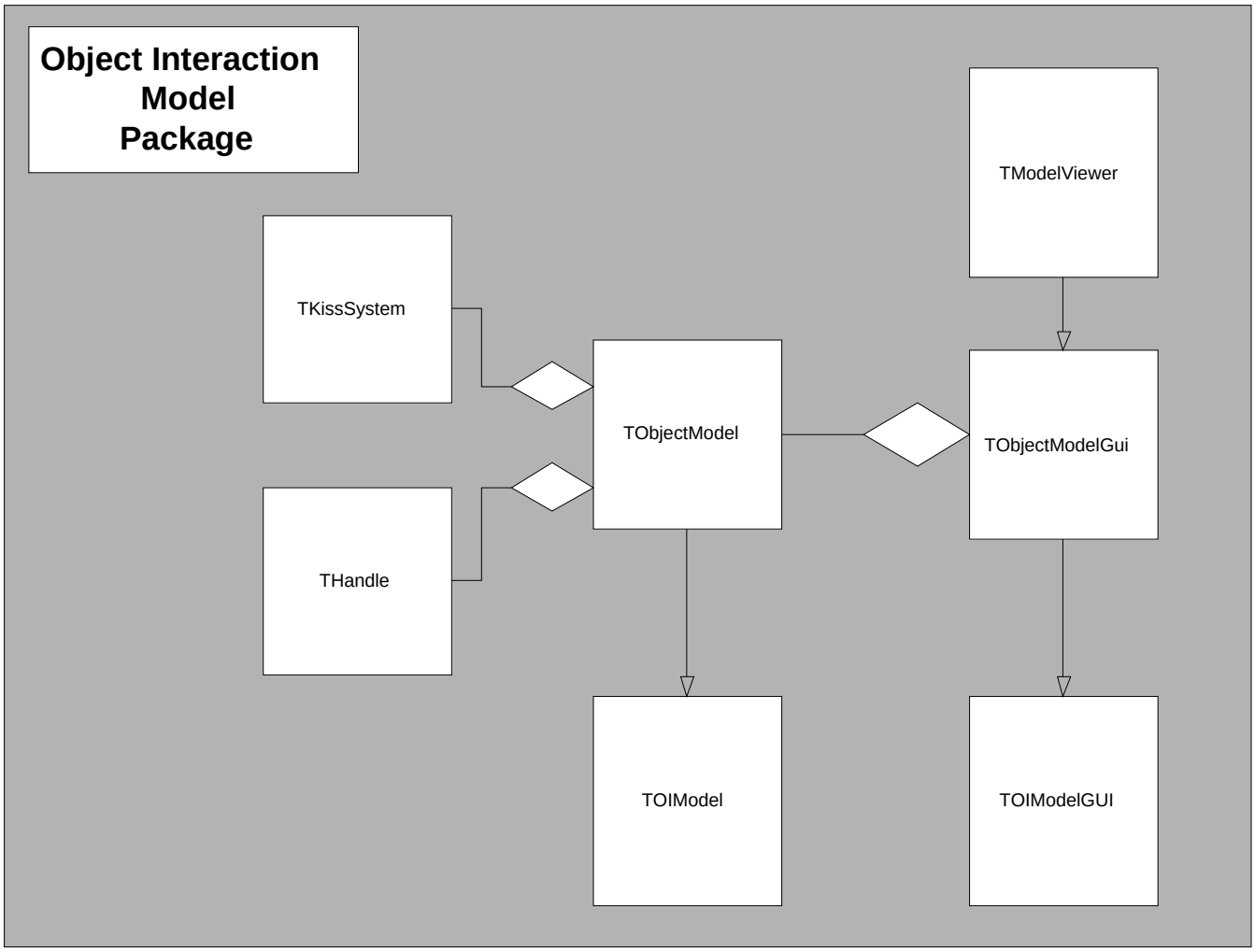
Date: 12-12-2001
Subject: Object InterAction Model Technical Documentation
Author: Sander Groenen

Content

1. Object interaction model description

The object interaction model is a model used by the KISS development language to view the objects within a software system and how they are related to each other. The interaction model in the kiss generator program is used to display all current objects in the kiss generator system. The model gets its data from a previously designed component “TKISS SYSTEM” and uses the methods for displaying of another previously designed component named “TMODELVIEWER”, for more detailed information on these component, see also “Model viewer Technical Documentation” and “Kiss System technical documentation”. For now it is sufficed to say that the Kiss System has all the data about objects within the kiss generator system and the Modelviewer was designed to create shapes and lines that have an affection with design models and to display them on screen. The two components namely: Object interaction model and Modelviewer seem to be very much alike and in a way they also are, but the difference between them is that Modelviewer is a generic component to display any kind of model one would like and the Object interaction model is specifically designed to display the object interaction model, using the modelviewer methods, of the current kiss generator system, as was stated above. The Object interaction model consists of a number of classes, which aggregate or specialize each other. Below is the figure that shows how these classes are related and following will be a description of each seperate class.



2. Object interaction Desgin

The object interaction model as shown in the figure above consists of 7 classes namely Tmodelviewer,TkissSystem,Thandle, TobjectModel, TobjectModelGui, TOIModel and TOIModelGui. These classes will be explained here, with the exception of TModelviewer and TKissSystem which can be found in “Model viewer technical documentation” and “Kiss system Technical documentation”. Another thing that has to be pointed out is the separation of the interaction model’s functionality(meaning data gathering) and GUI(drawing of the model), this was done by creating two base classes namely TobjectModel(functionality) and TObjectModelGUI(GUI) These two functions have been separated intentionally, because by doing that they can both be changed more easily. The communication of the two classes was realized by using the OBSERVER design pattern.

2.1 Thandle

Thandle is a class which is aggregated in the object model class. This Thandle class was designed only to make the insulation of the GUI classes possible, meaning that the GUI classes cannot be allowed to be aware of the specifications used by kiss system to represent data. To make it all a little cleare, here's an example: the kiss system uses the class TkissObject to represent an object within the system, if the GUI classes knew about the existence of TkissObject then there was no insulation and the seperation of functionality and GUI would be lost, but by creating a Thandle class in which a TkissObject can be inserted, the GUI classes only would have to know about that Thandle class and therefore only need to know about their data model classes such as TobjectModel(described below) and thus insulation is preserved.

2.2 TobjectModel

The object model class is a class that serves as a base for all models within the Kiss generator system, currently there are two models in this system, namely: the object interaction model and the kiss model, but in the future there may perhaps be more models added to the system and this class can also serve then as a base for this new models.
The object model functionality consists of methods to identify Thandle instances, because Thandle by itself does not represent any form of data. These methods translate the Thandle instance to a enumeration type defined the class itself(TobjectModel). An example: the following method declaration is one of the methods in the object model:

TElementType GetElementType (THandle) const;

This method is used to identify the handle instance. It returns a variable of the type TelementType. This type is an enumeration, that consists of the following set(etAction, etObject, etAssociation, etCombinator). So this method can tell what kind of data resides in the handle instance.The object model does know about data representations in the kiss system(a pointer to it, as shown in the figure by the aggregation symbol) in order to translate them to their own enumeration types. Note that the subtype "etAction" , which will be used by the GUI classes, has nothing to do with any type within KissSystem and therefore keeping the insulation intact(see paragraph 2 and 2.1).,also the object model is only used to gather all data about the objects within the kiss generator system(paragraph 2).

2.2 TOIModel

The TOIModel is a specialized form of the TobjectModel. It adds the functionality to gather data that is specifically needed to create a Kiss object interaction model. It uses the methods of the object model which identify the handle instances to filter out the needed data. Where the object model only has methods to identify the data, the Toi model actually retrieves the data and also has the possibility to change that data in the kiss

system. Another example may clarify these statements; the following is a method of the Toi model class:

TElementList GetElements();

This method uses the functionality of the kiss system to go through all the data in the kiss system and place all “needed” data in a Telementlist, which is a type def for a Thandle list. “needed data” means the data needed for the Toi model, because the data for example could exist of a Kiss combinator type, which is not needed for the Toi model but for the Kiss Model. Another method of the Toi model is:

void setTotality (THandle h,unsigned int i);

This method can set the totality(the maximal number of instances of an object associated with another) of an association(relation between objects) within the Kiss system, provided that the handle given to the method is an association type.

2.3 TObjectModelGui

The object model gui class is a class that serves as a base for all models within the Kiss generator system, currently there are two models in this system, namely: the object interaction model and the kiss model, but in the future there may perhaps be more models added to the system and this class can also serve then as a base for this new models.
The class is derived from the Tmodelviewer class(see Model viewer technical documentation) to use its functionality. The object model GUI class aggregates the object model class and uses this class to get all the data of the kiss system needed for the drawing of the object interaction model.
The object model gui does not actually draw the model, but provides functionality to store this data in its own lists. The preceding statement requires some explanation; while the object model en the Toi model provide the means of retrieving data from the Kiss System and inserting them into Thandles and also provides methods for identifying these handles(etAssociation, etAction etc..), the object model gui needs to associate these handles with a representation that the modelviewer can understand, namely a TabstrShape or a TabstrLine class. Therefore the object model gui maintains four lists(besides the lists that are within the modelviewer) to associate the handles with these classes.

The first two lists are the following, one to associate a handle with a TabstrShape the second to associate a TabstrShape with a handle. The second two list are the following, one to associate a handle with a TabstrLine, the second to associate aTAbstrline with a handle. The methods the object model gui provides for these lists are therefore the following:

void AddElement(THandle);
void AddElement(THandle, const TAbstrShape&);
void AddElement(THandle, TAbstrLine*);
DeleteElement(THandle);

LookupShapeRes LookupShape(THandle) const;
LookupLineRes LookupLine(THandle) const;
THandle LookupHandle (TAbstrShape*)const;
THandle LookupHandle (TAbstrLine*) const;

The Add element and Delete element methods add the elements to these four lists(depending on which form is added(TabstrShape or TAbstrline)) and also to the lists of the modelviewer. The lookup functions provide means of retrieving a TAbstrShape or TAbstrLine from a handle and vice versa.

2.4 TOIModelGui

The TOIModelGui is derived from the TobjectModeGui as is shown in the figure. The ToiModelGui uses the functionality implemented by the its parent(TObjModelGui) and its parent's aggregate(an instance of TobjectModel) to gather the data needed to actually draw the object interaction model(as was described in paragraph 2.3 the object model gui does not actually retrieve the data it only provide the means to store the data in the four lists).The following will explain this:

In the TOIModel the following method was provided to retrieve the elements from the Kiss System and put them into the Thandle instances:

TElementList GetElements();

In the Tobject Model Gui the following functions were provided to add and delete elements to the four lists associating Thandles with Tabstrshapes and TabstrLines:

void AddElement(THandle);
void AddElement(THandle, const TAbstrShape&);
void AddElement(THandle, TAbstrLine*);
DeleteElement(THandle);

The TOIModelGui(*see paragraph 2.6 for a note) has the following method:

void DrawModel();

This method uses the method “GetElements” from the TOIModel to retrieve the data from the KissSystem and then uses the AddElement methods from the TobjectModel to add the data to the four lists an the lists of the modelviewer and finally it uses the functionality of the modelviewer to draw the shapes and lines that were added on the screen.

Except for drawing all objects and associations between them in the kiss generator system, it also adds some means to react to user response, such as changing the name of an object or the plurality, connectivity and totality of an association. This is accomplished by a pop up menu when the user clicks with the right mouse button near an object or an association.

2.5 Note on TOIModelGui

There is some detail of the previously given figure that has to be explained. As was shown in that figure the TobjectmodelGui aggregates the TobjectModel. Also the TOIModel gui was derived from the TOBjectModelGui and therefore should have an aggregated instance of TobjectModel within it, but actually it has an instance of ToiModel . This is done at the creation of a ToiModelGui instance, at that time the TobjectModel instance is replaced by a ToiModel instance, which is also a TobjectModel form because it is derived from that very same class. That is why the TOIModelGui can acces all of the methods of the TOIModel and so the relationship between these two classes, which at first sight do not seem to do anything with each other, is explained.