ARIS UML Designer - Introduction

ARIS UML Designer
Version 9.8 – Service Release 2

October 2015

Product Documentation

Contents

1 Introduction

This document provides an overview of the key functionalities of ARIS UML Designer 9 and outlines the underlying concepts. It is aimed at all UML modelers, developers of UML-specific reports and macros, and ARIS administrators. If you are not yet familiar with ARIS, you should first read the document **ARIS Architect Quick Start Guide**, as basic knowledge of ARIS is essential to understand this introduction.

This introduction is divided into the following sections:

UML Basics: Contains a brief introduction to UML and introduces the diagram types supported by ARIS UML Designer. This section also sets out the basics of the UML metal model, which is an essential requirement for you to understand the subsequent topics of **Mapping UML to the ARIS object model** and **UML profiles**.

ARIS UML Designer overview: Introduces the most important components of ARIS UML Designer and their functionality.

Mapping UML to the ARIS object model: Shows how UML content is stored in ARIS so that you can understand how ARIS standard functions, such as definition copy, merge or access privileges affect UML content.

Linking business process and UML modeling: Outlines the relevant technical principles.

UML profiles: Provides an overview of the extension and filter mechanisms in UML, which differ fundamentally from the classic ARIS method configuration.

Differences from ARIS Architect and ARIS Designer: Shows the special features of ARIS UML Designer and the resulting differences from classic ARIS modeling.

Differences to ARIS UML Designer 7.x: Deals with the new features of ARIS UML Designer 9 compared to the previous version 7.x.

As the terms **model** and **object** that are familiar from ARIS have a different meaning in UML, the terms **diagram** and **element** are used to refer to them in this document. Thus, in this document an EPC is not a model but a diagram, and an EPC function is not an object but an element.

ARIS UML Designer 9 supports UML version 2.5. For compatibility reasons, ARIS Architect and ARIS Designer still support the old UML version 1.4. However, this is not done using UML-specific notation but in the form of classic ARIS diagrams. Where the term **UML** is used without a version number in this document, it always refers to the version 2.5 supported by ARIS UML Designer.

2 UML Basics

This section contains a brief introduction to UML and the underlying metamodel, where this is necessary to understand the remaining sections of this document and to start using ARIS UML Designer. For detailed information on UML and the individual UML element and diagram types, refer to the UML specification itself and to relevant secondary literature. In addition, basic knowledge of object-oriented principles are very useful in understanding UML.

2.1 What is UML?

UML stands for **Unified Modeling Language** and is a modeling language for object-oriented (software) systems. The word **software** is in brackets here as the design and description of software systems was definitely the focus in the development of UML, in theory UML is actually suitable for modeling any systems using an object-oriented perspective. One example would be describing the IT-related aspects of a business domain as part of an object-oriented analysis, in order to derive requirements for a corresponding software system.

UML was presented in the mid-1990s by Grady Booch, Ivar Jacobson, and James Rumbaugh as a joint development of their own object-oriented methods (Booch method, OOSE, and OMT). Since 1997, UML has been published as a standard by the OMG and has been continuously developed. The OMG (Object Management Group) is a non-profit organization that is responsible for a range of important standards in the IT industry. In addition to UML, these include BPMN.

The current UML version at the time of this document's creation is 2.5. This is also the version supported by ARIS UML Designer 9. The corresponding UML specification is available from the OMG at the following address: http://www.omg.org/spec/UML/.

2.2 UML diagram types

UML differentiates two categories of diagrams - structure diagrams and behavior diagrams. In structure diagrams, the focus is on static structures. One example is the Component diagram, which is used to model the relationships between individual (software) components. By contrast, there are behavior diagrams, which focus on the dynamic behavior of a system. An example of a behavior diagram would be the State machine diagram, which shows how the instance of a class changes its internal state when particular events occur.

A typical feature of structure diagrams is that they show a selected section of a UML structure. They represent a view of a freely definable subset of this structure. By contrast, behavior diagrams (apart from the Use case diagram) describe the dynamic behavior of a particular element, normally a state machine, an activity, or an interaction.

UML defines the following diagram types:

Structure diagrams	Behavior diagrams
Class diagram (Class diagram)	Activity diagram (Activity diagram)

Component diagram (Component diagram)	Use case diagram (Use case diagram)
Composite structure diagram (Composite structure diagram)	Communication diagram (Communication diagram)
Object diagram (Object diagram)	Interaction overview diagram (Interaction overview diagram)
Package diagram (Package diagram)	Protocol state machine diagram (Protocol state machine diagram)
Profile diagram (Profile diagram)	Sequence diagram (Sequence diagram)
Deployment diagram (Deployment diagram)	Timing diagram (Timing diagram)
	State machine diagram (State machine diagram)

The individual diagram types are briefly introduced below using simple examples relating to the topic of **order processing**. These examples are not intended to provide a complete or technically correct representation. They are merely intended to illustrate how particular situations relating to the topic can be represented using the different UML diagram types.

2.2.1 Class diagram

Class diagrams primarily show relationships between classes and their properties.

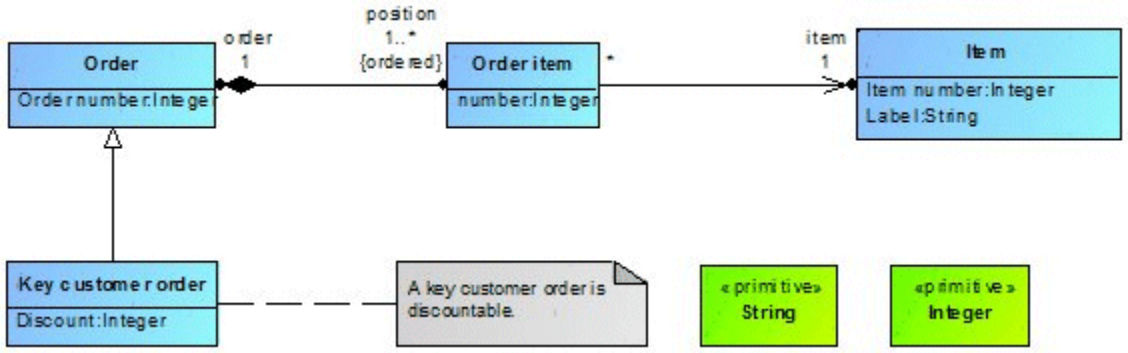


Figure 1: Class diagram

Figure 1 shows a simple Class diagram with four classes (UML type **Class**) **Order**, **Key customer order**, **Order item**, and **Item**, two primitive data types (UML types **PrimitiveType**) **String** and **Integer**, and a comment (UML type **Comment**). The classes each contain one to two attributes (UML type **Property**). For example, the **Order** class contains the **Order number** attribute, which is of the **Integer** type. This means that the order number is a property whose value is a whole number.

UML itself does not stipulate which data types are to be used when modeling and how they should be denoted[1]. In the example above, the two types **String** and **Integer** could also be called **Character string** and **Whole number**.

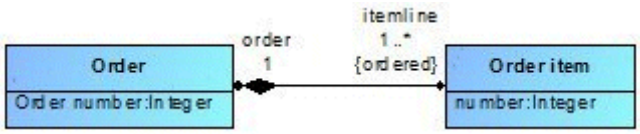


Figure 2: Association between two classes

The **Order** and **Order item** classes are linked together by an association. This association has an association end **order** with the multiplicity **1** and an association end **itemline** with the multiplicity **1..***. The multiplicity **1..*** at the **itemline** association end means that at least one and up to any number of order items are assigned to an order. The property **{ordered}** specifies that the order items are assigned to the order in a particular sequence. The multiplicity **1** at the **order** association end means that an order item is always assigned to only one order.

The shaded black diamond on the association indicates that the order items are part of the order and if an order is deleted its order items will also be deleted automatically.

[1] One exception is the modeling of profiles in ARIS UML Designer, where the corresponding types specified by UML have to be used for primitive data types.

The two black dots at the end of the association edge mean that the corresponding association ends are simultaneously attributes of the respective opposing class. Figure 3 therefore shows a semantically equivalent alternative appearance of Figure 2[2].

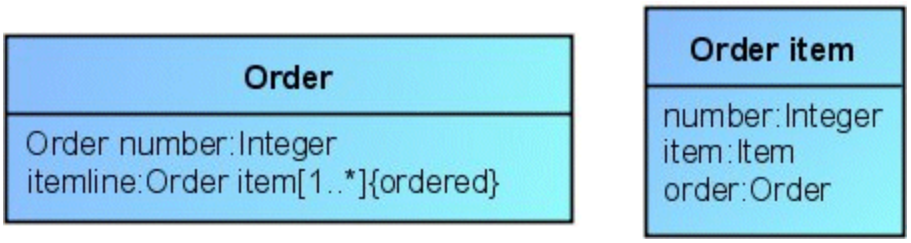


Figure 3: Association ends as attributes

An arrow head at an association end means that the association is navigable in the direction of the arrow. Figure 4 shows an association that is only navigable in one direction.

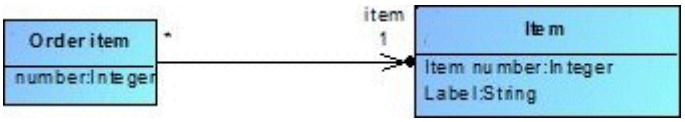


Figure 4: Unidirectional association

In this example, the order item knows which item it relates to. Conversely, the item has no knowledge of the order items by which it is used.

An association end is classed as navigable if it is simultaneously an attribute of the opposite class[3]. If both ends of an association are navigable, there is no need for you to display the two arrows in the diagram (see Figure 2).

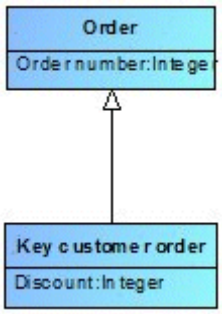


Figure 5: Generalization between two classes

Figure 5 shows a generalization relationship (UML type **Generalization**) between the **Key customer order** and **Order** classes. This means that the **Key customer order** class is a specialization of the **Order** class and thus inherits all properties from it.

[2] The **order** attribute for the **Order item** class is displayed within the class without specifying its multiplicity **1** as the value **1** is the default value for multiplicities and it is not normally displayed within classes.

[3] There is another method of specifying the navigability of an association end. However, it will not be discussed here.

2.2.2 Component diagram

Component diagrams show relationships between components and their properties.

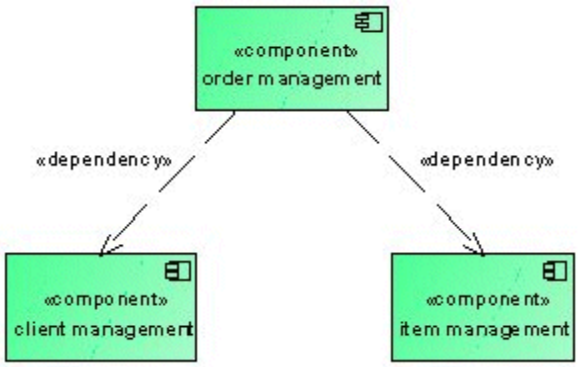


Figure 6: Component diagram

Figure 6 shows the simplest form of a Component diagram. It shows three (software) components and their dependencies. From it, we can determine that the components for client and item management can work independently of other components, while the **order management** component has dependencies on the other two components. This diagram provides no information about the nature of these dependencies.

The component dependencies can be described in more detail by using ports and interfaces. Figure 7 shows an example of this.

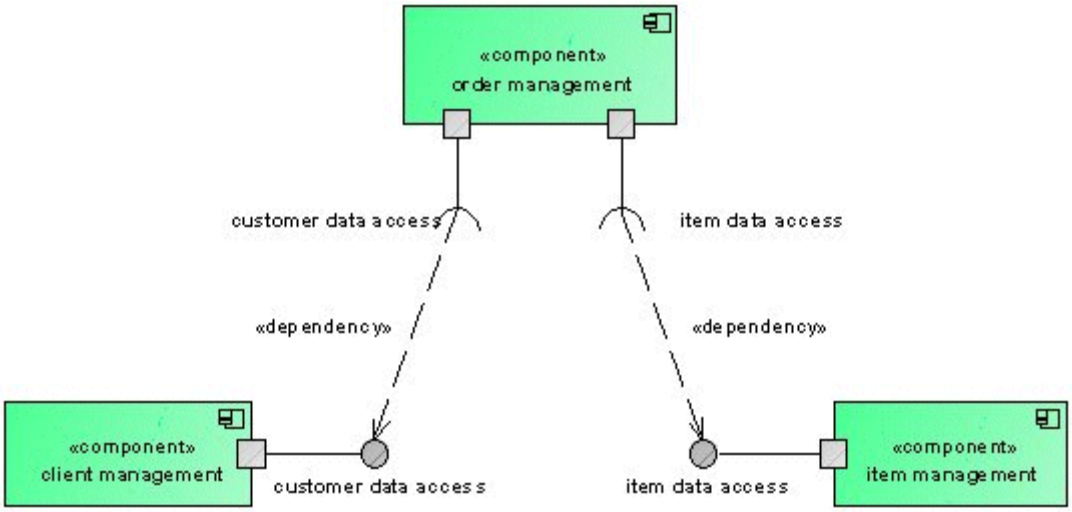


Figure 7: Components with ports and interfaces

The **client management** component uses a port to provide an interface called **customer data access**, which can be used to retrieve customer data from the component. Likewise, the **item management** component also provides a corresponding interface called **item data access**. In turn, the **order management** component uses two ports to specify that it requires access to the **customer data access** and **item data access** interfaces. Two dependency relationships illustrate the access to the interfaces.

2.2.3 Composite structure diagram

Composite structure diagrams show the internal structure of a class and the relationships of the individual class components to one another.

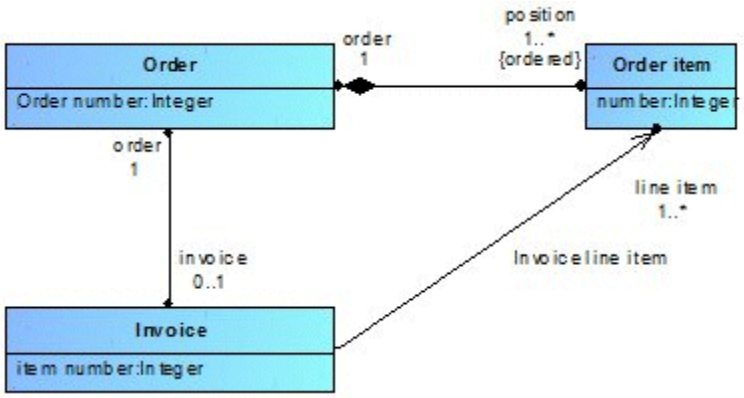


Figure 8: Class diagram with order components

Figure 8 extends the example from Figure 2 with an additional class involved in an order, the invoice. Unlike the order item, the invoice is not a compositional component of an order but is directly assigned to it. The invoice also refers to the individual order items.

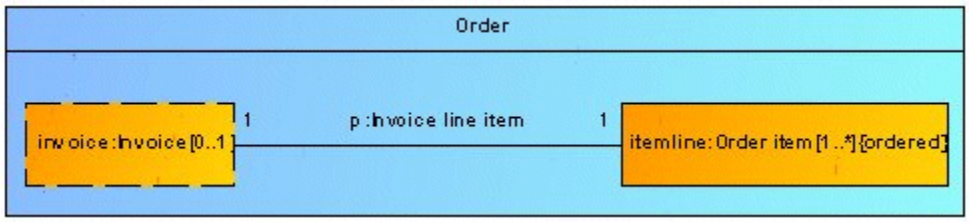


Figure 9: Composite structure diagram for the **Order** class

The Composite Structure diagram in Figure 9 provides an alternative representation to the class diagram in Figure 8. The two attributes **invoice** and **itemline** are represented graphically in the **Order** class as symbols of the **Property** or **Part** type. The dotted border of the **invoice** property symbol indicates that the invoice is not a compositional component of the order, while the solid border of the **itemline** part symbol means that order items are included in the composition of the order.

The two symbols are linked by a connector with the name **p**, which refers to the **Invoice line item** association shown in Figure 8.

2.2.4 Object diagram

Object diagrams show the relationships between instances of different classes.

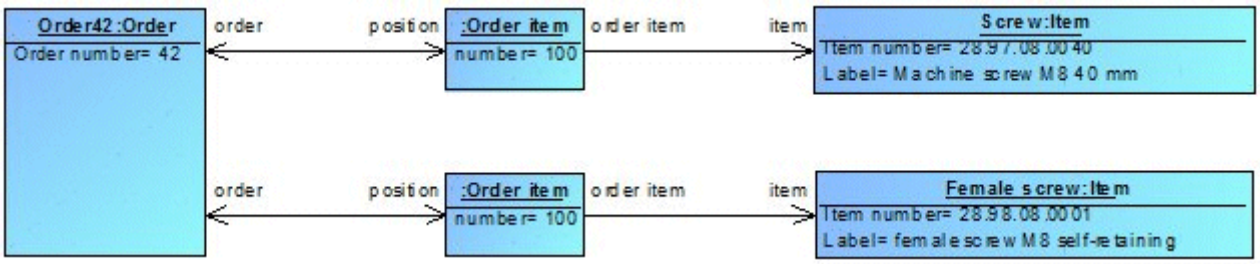


Figure 10: Object diagram with order instance

Figure 10 shows a specific order with two order items.

The order, the two order items, and the two items are instances of the corresponding classes from Figure 1. Their relationships represent instances of the associations between these classes.

Instances of classes are also referred to as objects and instances of associations as links. However, all instances are technically of the UML type **InstanceSpecification**[4].

Instances can optionally show their own name and/or the name of their class, separated by a colon. In this example, the order object and the two item objects show both their own names and the name of their class, while the two order item objects only show their class names.

The attribute values for the objects are of the UML type **Slot**. A Slot shows the name of an attribute of the class for the instance (e.g., **Order number**) and the value that this attribute has for the instance (e.g., **42**). The link ends (e.g., **position**) are also of the **Slot** type. However, as they are directly linked to their value, i.e., the corresponding object, in the diagram, only the relevant attribute name is displayed in the diagram.

[4] UML 2 only recognizes one **InstanceSpecification** type for instances of all classifier types, whereas UML 1.4 defined a special instance type for each classifier type (e.g., Class -> Object, Association -> Link, AssociationClass -> LinkObject, Component -> ComponentInstance etc.).

2.2.5 Package diagram

Package diagrams show package hierarchies and dependencies between packages.

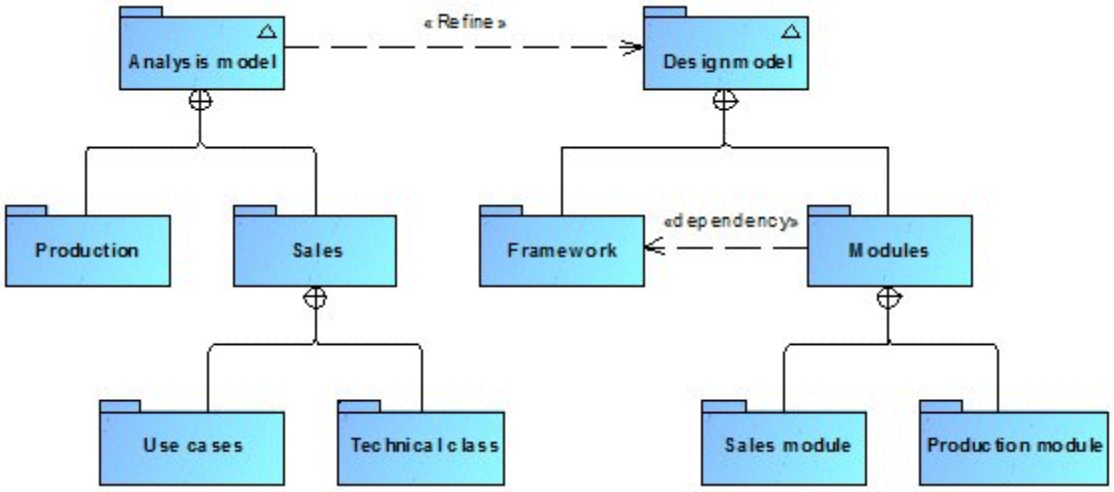


Figure 11: Package diagram

Figure 11 shows a Package diagram with two package hierarchies and dependency relationships.

At the top level it contains the two models **Analysis model** and **Design model**. Unlike in ARIS, in UML the term **model** does not refer to a diagram, but to a view of a physical system in a defined context. A model normally contains a whole series of elements, relationships, and diagrams, which all combine together to describe the model.

In line with this definition, in this example the analysis model represents an object-oriented view of the business processes to be supported by a new piece of software to be developed. This is used to derive a design model, which specifies the architecture and the individual modules of the software in more detail. This derivation is represented by a Refine relationship in the diagram. This relationship is an element of the UML type **Abstraction** with the stereotype **«Refine»**.

Within the design model, the dependency relationship between the **Modules** and **Framework** packages indicates that the Framework defines structures that are required by the modules.

2.2.6 Profile diagram

Profiles represent user-defined extensions or restrictions of the UML metamodel. They can be created using Profile diagrams.

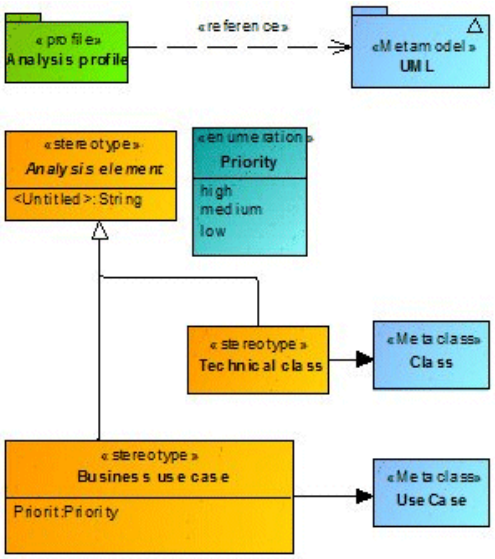


Figure 12: Profile diagram

Figure 12 shows a Profile diagram for defining a simple profile for object-oriented analysis.

On the left-hand side, the diagram contains the **Analysis profile**, which defines the two stereotypes **Technical class** and **Business use case**. Both of these inherit from an abstract Analysis element stereotype, which defines the **Contact** attribute. In addition to the stereotypes, the profile also defines an enumeration called **Priority**, which is used as a type by the attribute of the same name for the **Business use case** stereotype.

The right-hand side of the diagram contains the UML metamodel and two of its metaclasses. The profile has a relationship of the **Metamodel reference**[5] type to the metamodel. This relationship is always necessary if all UML content is to be available within packages to which the profile is assigned.

The two stereotypes **Technical class** and **Business use case** have a relationship of the UML type **Extension** with the **Class** metaclass or the **UseCase** metaclass. This relationship specifies the UML elements to which the stereotype can be assigned.

The attributes of a stereotype are available as additional properties for the UML elements to which the stereotype is assigned. In this example, a use case with the **«Business use case»** stereotype has the additional properties **Contact** and **Priority** in addition to the properties defined by the UML specification.

The 6UML profiles section contains more information about **profiles and stereotypes**.

[5] Strictly speaking, it is a relationship of the UML type **PackageImport**, which is referred to as a metamodel reference in this context and is also displayed in the graphical view with the keyword **«reference»** instead of **«import»**.

2.2.7 Deployment diagram

Deployment diagrams show the assignment of software components to physical IT systems and the networking of these systems with one another.

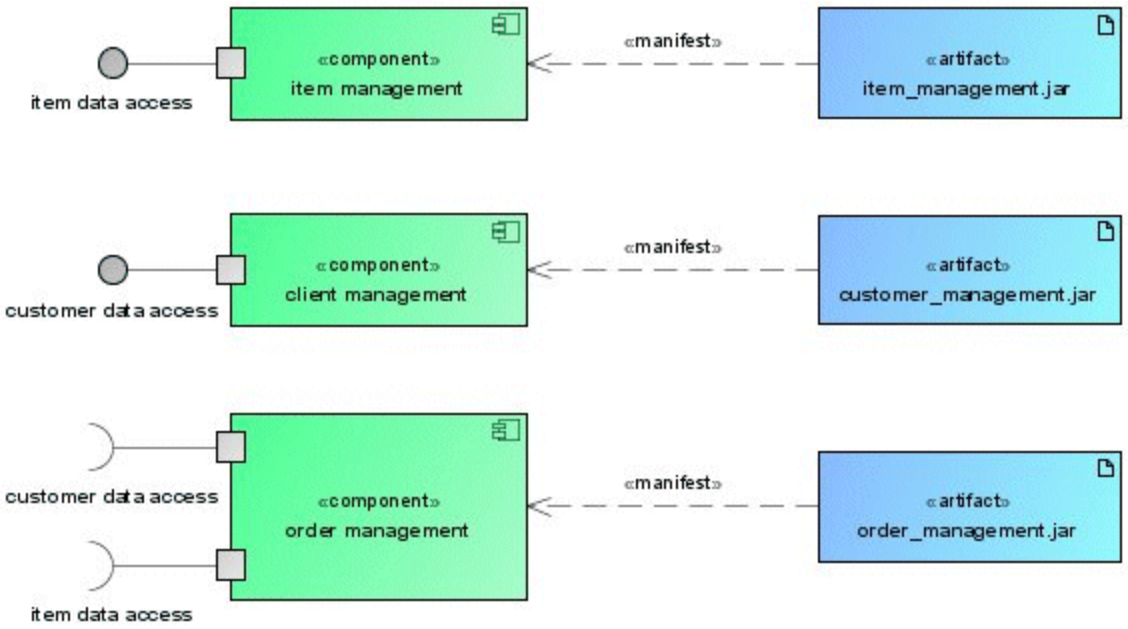


Figure 13: Deployment diagram with components and artefacts

Figure 13 shows the physical manifestation of the components from Figure 7 as JAR files[6]. This is done using UML elements of the **Artefact** type, which are linked to the corresponding components by a relationship of the **Manifestation** type.

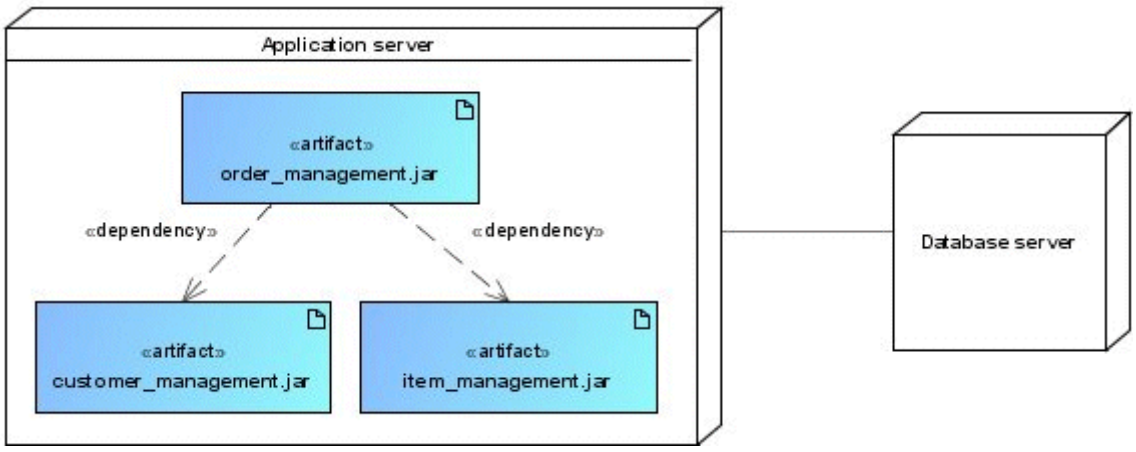


Figure 14: Deployment of software components on physical systems

Figure 14 shows how the artefacts or software components defined in Figure 13 are deployed on physical systems and the relationships between these systems.

[6] Java libraries

2.2.8 Activity diagram

Activity diagrams show dynamic processes in the form of a graph of individual actions. They can describe both a process with a high level of abstraction or details of an algorithm in a piece of software.

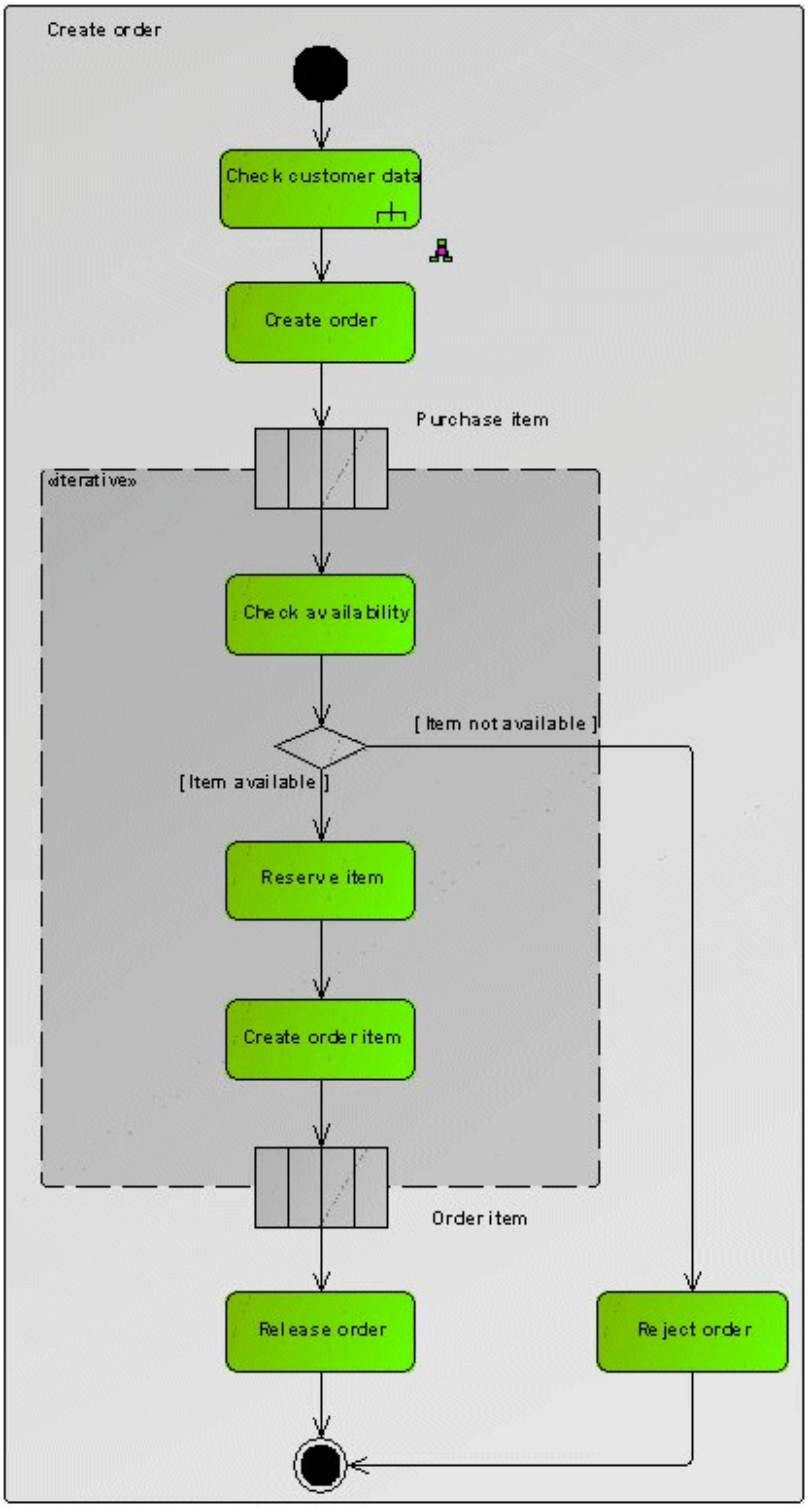


Figure 15: **Create order** Activity diagram

Figure 15 shows the process for creating an order in the form of a UML Activity diagram. The activity starts with an element of the **InitialNode** type and ends with an element of the **ActivityFinal** type.

The first step is to check the customer data. This is done using an action of the **CallBehaviorAction** type. This is an action that in turn invokes an activity **Check customer data**. This is described in a further Activity diagram (see Figure 16).

The other actions are all of the **OpaqueAction** type. They are characterized by the fact that they are specified by a simple text and have no further UML semantics.

The individual purchase or order items are processed in an ExpansionRegion. This has the customer's purchase items as its input elements and the corresponding order items as its output elements. The ExpansionRegion is run through for each input element.

The relationships in the two Activity diagrams are all of the **ControlFlow** type. If a text in square brackets is specified on one of these control flow edges, it describes the condition that has to be met for the control flow to proceed along this edge.

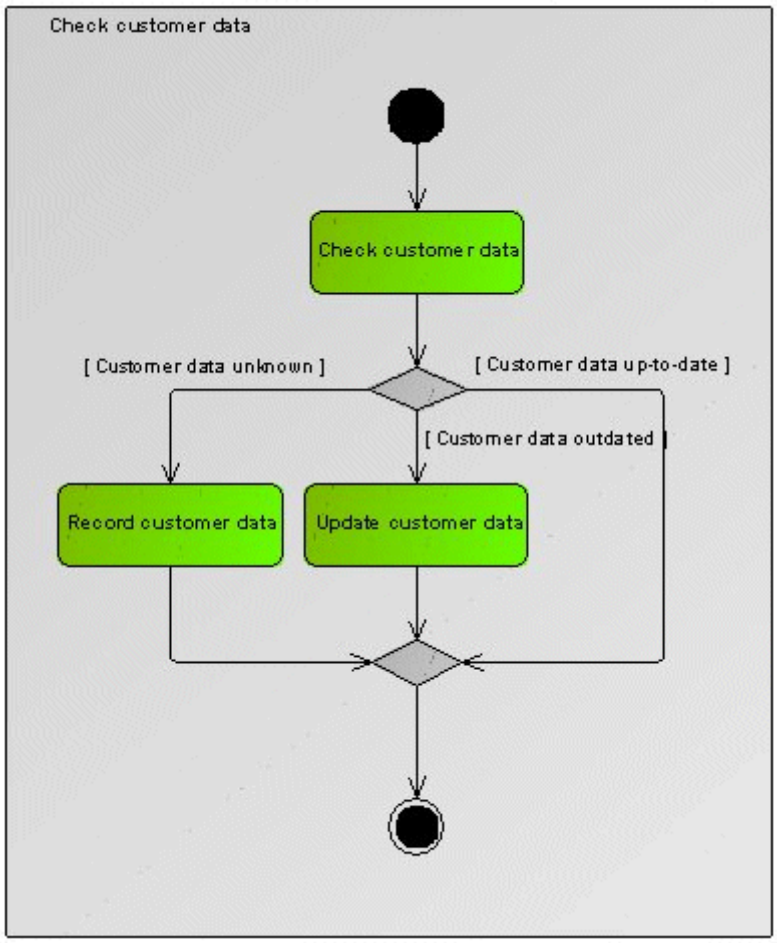


Figure 16: **Check customer data** Activity diagram

2.2.9 Use case diagram

Use case diagrams are used as part of an object-oriented requirements analysis to describe the use cases to be analyzed and the actors involved.

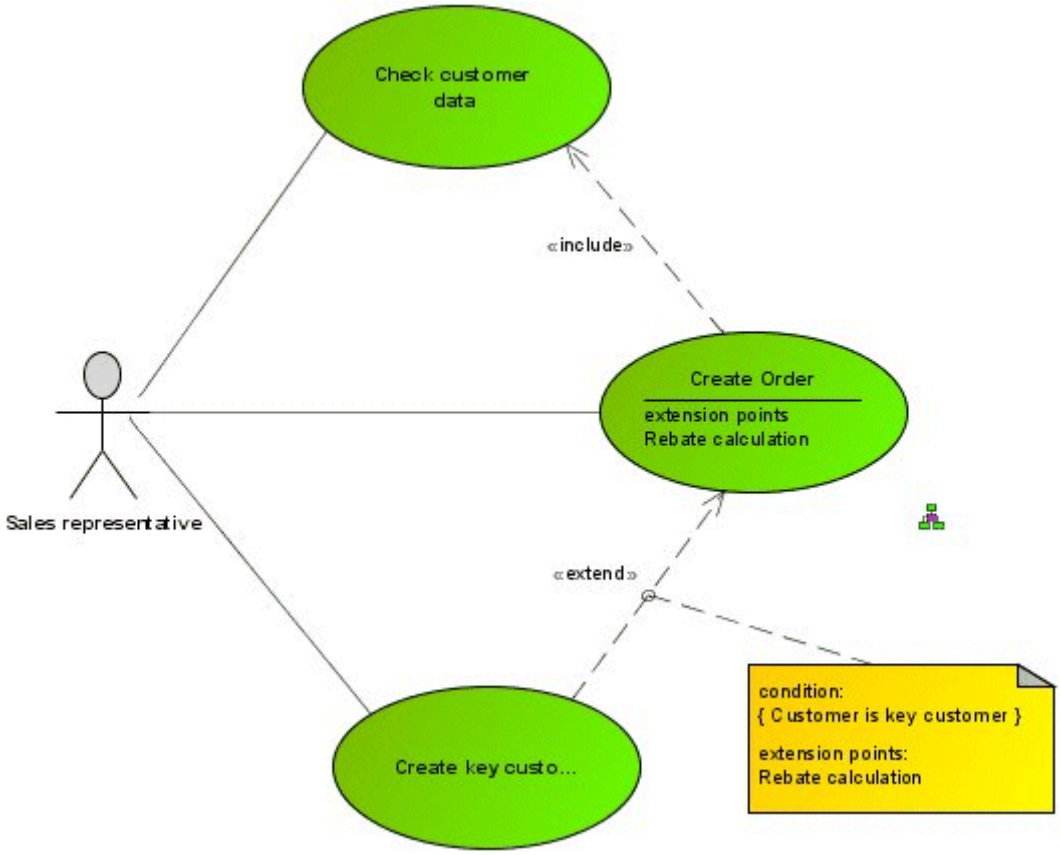


Figure 17: Use case diagram

Figure 17 shows a Use case diagram with one actor and three use cases. Actors are people, roles, or systems that interact with the system to be analyzed. A use case represents a self-contained functionality that can be invoked from outside the system and leads to a particular result.

The relationships between the actor and the use cases are associations. In Use case diagrams, these are normally shown without annotations such as names or multiplicities.

A relationship of the **Include** type runs between the **Create order** and **Check customer data** use cases. This means that the **Create order** use case includes the **Check customer data** use case, i.e., when creating an order the customer data is checked.

The **Create order** use case defines an extension point called **Rebate calculation**. An extension point indicates a particular point in the internal process of the use case at which additional functionality can be added to the process. In this example, this is the calculation of a rebate.

The Extend relationship between the **Create key customer order** and **Create order** use cases states that the **Create key customer** use case supplements the **Create order** use case with this functionality at its **Rebate**

calculation extension point, in this case by calculating a special key customer rebate. In addition, the Extend relationship includes a specification of the condition under which the extension is permitted, namely that the customer must be a key customer.

Typically, the internal process of the use cases would be described using behavior diagrams, for example the two Activity diagrams in Figure 15 and Figure 16.

2.2.10 Communication diagram

Communication diagrams show how elements exchange messages with one another as part of an interaction. In contrast to Sequence diagrams, in Communication diagrams the focus is on the channels via which these messages are sent rather than on the chronological sequence of exchanging messages.

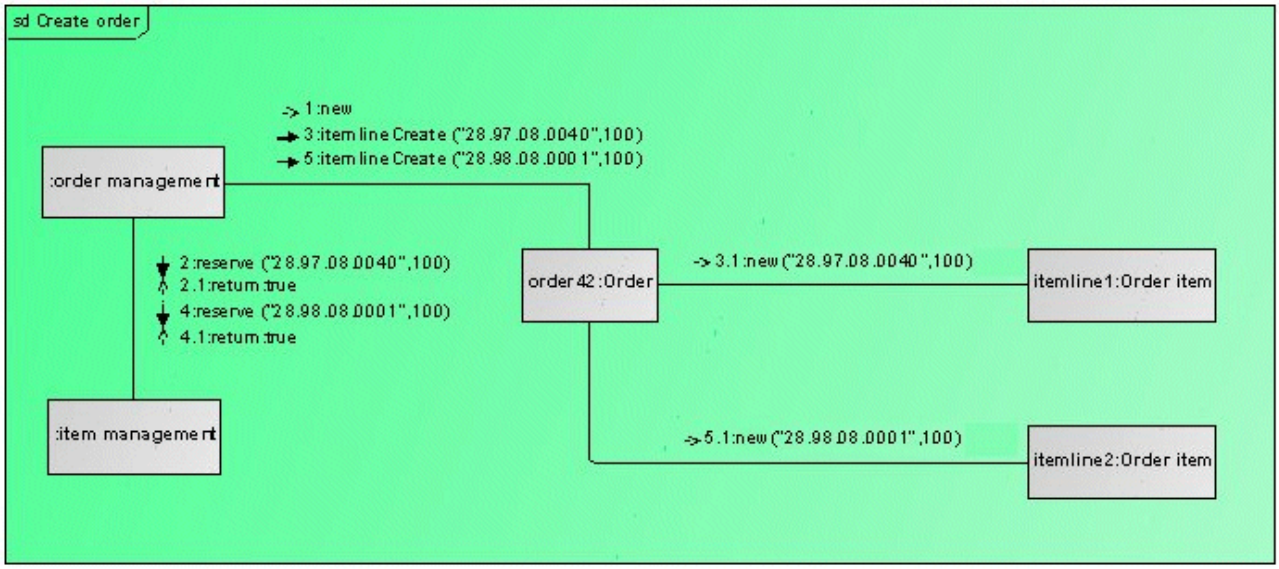


Figure 18: **Create order** Communication diagram

Figure 18 uses a Communication diagram to show how an order is created in the system. The background object that contains all other elements is the interaction itself. The contained elements are elements of the **Lifeline** type. The term **Lifeline** comes from its use in Sequence diagrams, where it is represented by a rectangle with a vertical line attached to the bottom.

The lifelines do not show their own names in the diagram, they show the name of the element represented and/or the type name of the element represented[7]. Depending on the context, the element represented can be an attribute (property), a port, or a parameter. In the above example a property has been created as an attribute for the interaction for the **:Order management** lifeline and the **Order management** component from Figure 6 has been assigned to it as the type. As the name of the attribute is irrelevant here, the lifeline only shows the type name.

The relationships between the lifelines are the connectors familiar from Composite structure diagrams. It is important to note that although they connect the lifelines with one another graphically in the diagram, they are actually relationships between the elements represented. This means that if no represented element is assigned to a lifeline, you cannot connect it to another lifeline in the diagram using a connector.

The messages that the lifelines exchange with one another along the connectors are displayed as text on the connector. An arrow specifies the direction of the message, and the order is given by the sequence numbers of the messages. If the messages have arguments, they are listed in brackets after the message.

[7] The element represented is assigned to the lifeline through its **represents** property.

This Communication diagram should be read as follows:

1: Order management creates a new order object.

2: Order management reserves 100 items with the item number 28.97.08.0040.

2.1: Item management confirms the reservation.

3: Order management instructs the order object to create a new order item for the item number 28.97.08.0040 with the quantity 100.

3.1: The order object creates a new order item with the corresponding parameters.

4. – 5.1: This process is repeated for another item.

The sequence numbers are automatically calculated in ARIS UML Designer based on the sequence of the messages on the source and target lifelines. If you want to change them, you have to do this in the Properties dialog for the relevant lifelines. Alternatively, you can hide the display of the calculated sequence numbers and assign your own numbers as part of the message name.

2.2.11 Sequence diagram

Sequence diagrams show how elements exchange messages with one another as part of an interaction. In contrast to Communication diagrams, in Sequence diagrams the focus is on the chronological sequence of exchanging messages rather than on the channels via which these messages are sent.

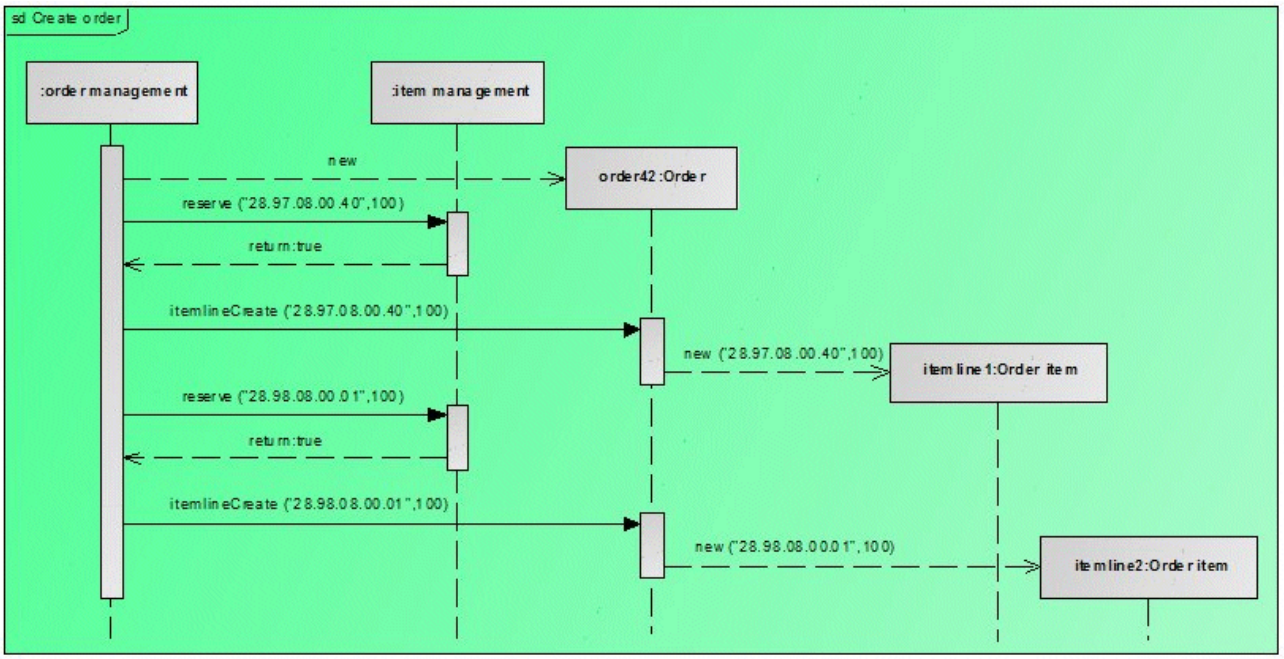


Figure 19: **Create order** Sequence diagram

Figure 19 shows the **Create order** interaction shown in Figure 18 in the form of a Communication diagram as a Sequence diagram. Both diagrams contain elements of the **Lifeline** type. However, in the Sequence diagram they have an additional dotted vertical line[8]. The messages are represented by graphical edges in the Sequence diagram.

While the chronological order of the messages is given by their sequence numbers in the Communication diagram, in the Sequence diagram this is defined by their vertical arrangement. The time axis, which is not explicitly visible, runs vertically from top to bottom in the diagram, i.e., messages closer to the top are transferred before messages located further down.

The rectangles on the lifelines are elements of the **ExecutionSpecification** type. They indicate when the object represented by the lifeline is active.

[8] The name **Lifeline** can be traced back to this type of graphical representation.

2.2.12 Timing diagram

Timing diagrams show how the state of the elements involved changes when exchanging messages as part of an interaction. The notation is based on corresponding diagrams from digital technology. This diagram type is primarily of interest for modeling of software systems closely related to hardware and has only rudimentary support in ARIS UML Designer 9.

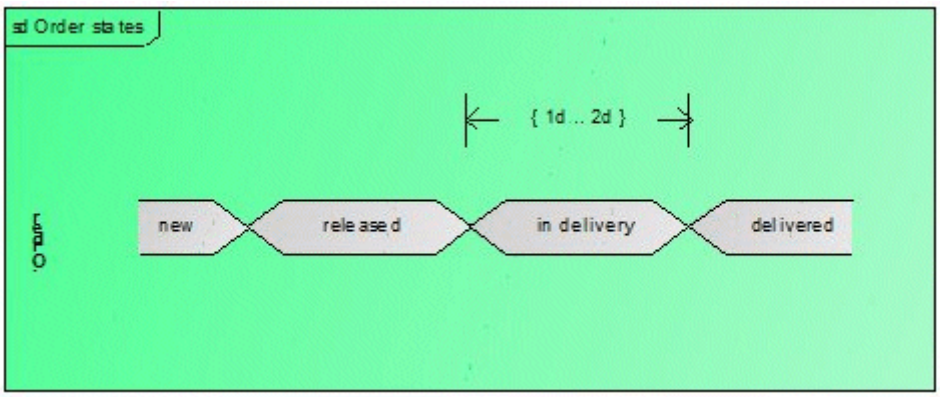


Figure 20: **Order states** Timing diagram

In Figure 20 a simple Timing diagram using compact notation[9] shows the states an order can have. The **Order states** element is an interaction, the **:Order** element is a lifeline, and the different states are represented using state invariants (UML type **StateInvariant**). The **DurationConstraint** element also specifies that the delivery state can only last for one to two days.

[9] The detailed notation with which the states and their transitions are displayed using timelines is not supported by ARIS UML Designer.

2.2.13 Interaction overview diagram

An Interaction overview diagram is part of a larger scenario and shows the sequence of individual interactions, which are normally modeled using Communication or Sequence diagrams.

These diagrams are Activity diagrams in which only actions of the **CallBehaviorAction** type are used, which invoke interactions.

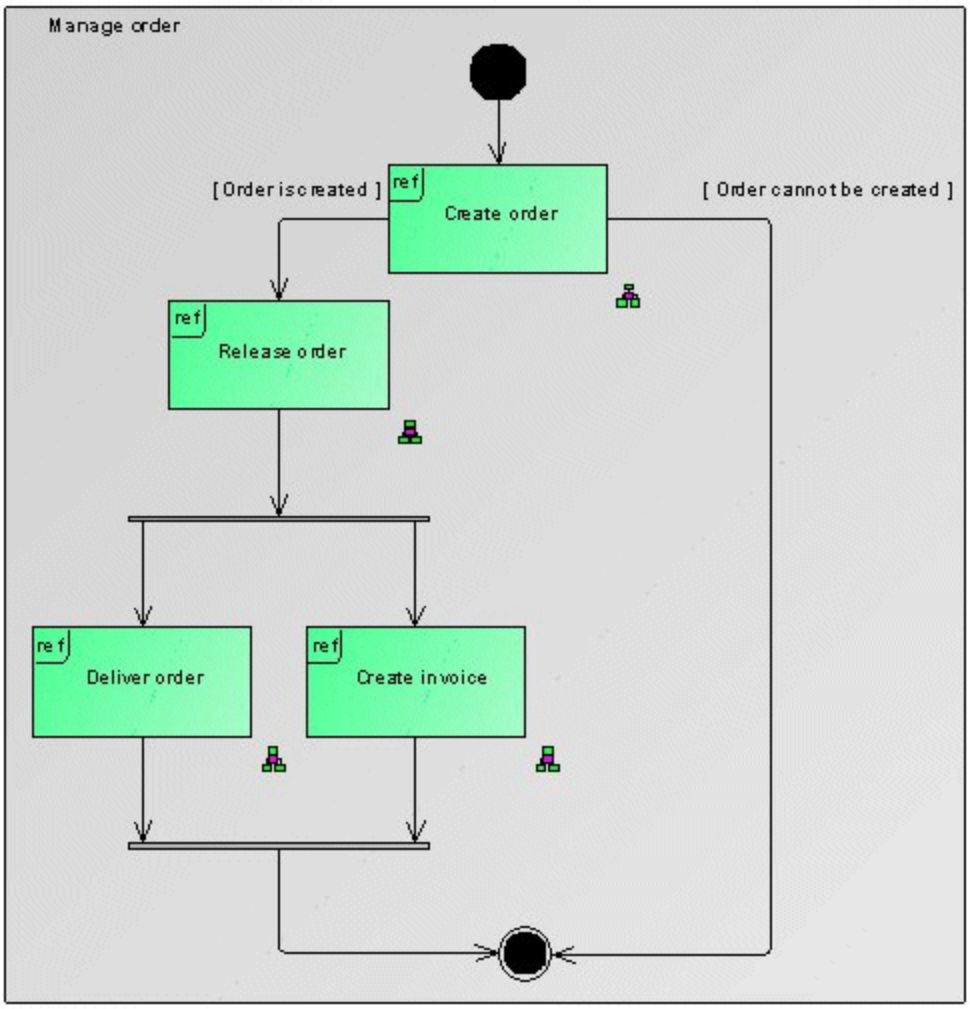


Figure 21: **Manage order** Interaction overview diagram

Figure 21 shows an Interaction overview diagram, which invokes interactions including the **Create order** interaction modeled in Figure 18. The individual actions of the **CallBehaviorAction** type each show the name of the interaction invoked.

2.2.14 State machine diagram

State machine diagrams show which states (UML type **State**) a system can take on, the relationship between them, and which events trigger the relevant status transitions (UML type **Transition**). A State machine diagram represents a state machine (UML type **StateMachine**). The states and their transitions are parts of this state machine.

The state machine can describe the behavior of a class (UML type **Class**), the behavior of another classifier of the UML type **BehavioredClassifier**, or the behavior of an operation. However, it can describe a behavior very generally without relating to one of these elements.

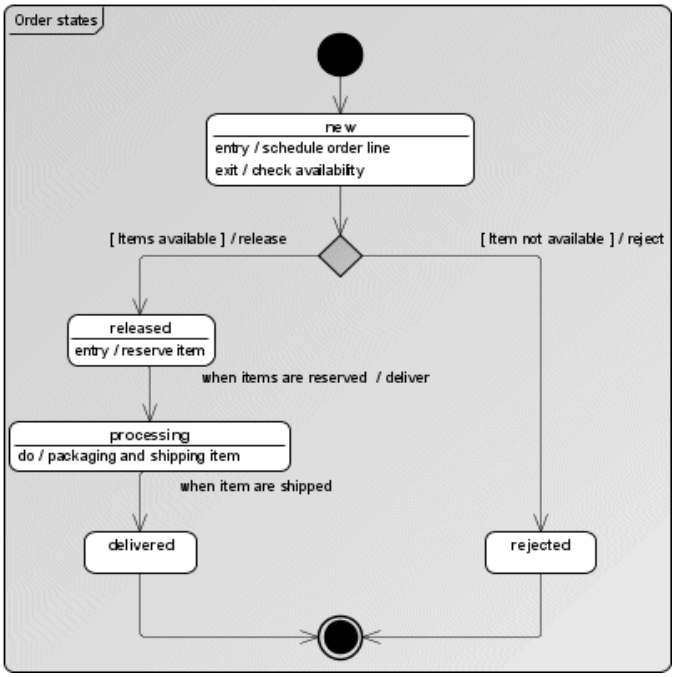


Figure 22: **Order states** State machine diagram

Figure 22 shows the states of an order. While the Timing diagram from Figure 20 only shows a sequence of state transitions, the State machine diagram includes the entire state machine with all states and their possible transitions as graphical edges, which link the states to one another.

The first state that an order can have is the **new** state. When this state occurs, the order items are recorded (**entry** property of the UML type **State**). An availability check for the items in the order items completes the state (**exit** property of the UML type **State**).

If a text is displayed in square brackets on a state transition, it refers to the condition (**guard** property of the UML type **Transition**) that has to be met for the state transition to occur. In the example diagram, these are the two conditions **[Items are available]** and **[Items are not available]**.

The text that follows the **/** character at a state transition refers to a behavior that is triggered by the state transition (**effect** property of the UML type **Transition**). In the example diagram, this would be **release**, **reject**, and **deliver**.

If a text at a state transition is neither in square brackets nor after a slash, it is the trigger (UML type **Trigger**) of the state transition (**trigger** property of the UML type **Transition**). In the example diagram, these are the two texts that start with the word **when**. The keyword **when** means that the triggers are assigned events of the UML type **ChangeEvent** (**event** property of the UML type **Trigger**).

2.2.15 Protocol state machine diagram

Protocol state machines (UML type ProtocolStateMachine) show the externally observable states and state transitions for a classifier. They define a protocol that the implementations of the relevant classifier must adhere to and show when, in what order, and under what conditions the publicly visible operations for that classifier are invoked.

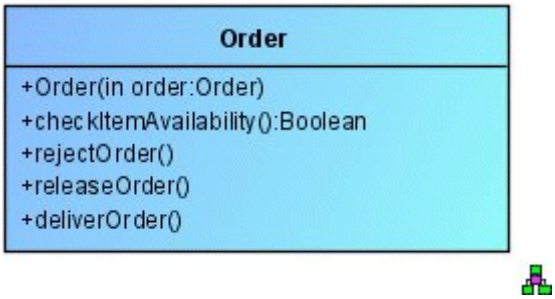


Figure 23: **Order** class with operations

Figure 23 shows the **Order** class with its public operations (UML elements of **Operation** type with the property **visibility = public**, in each case identifiable by the **+** character at the beginning of the name).

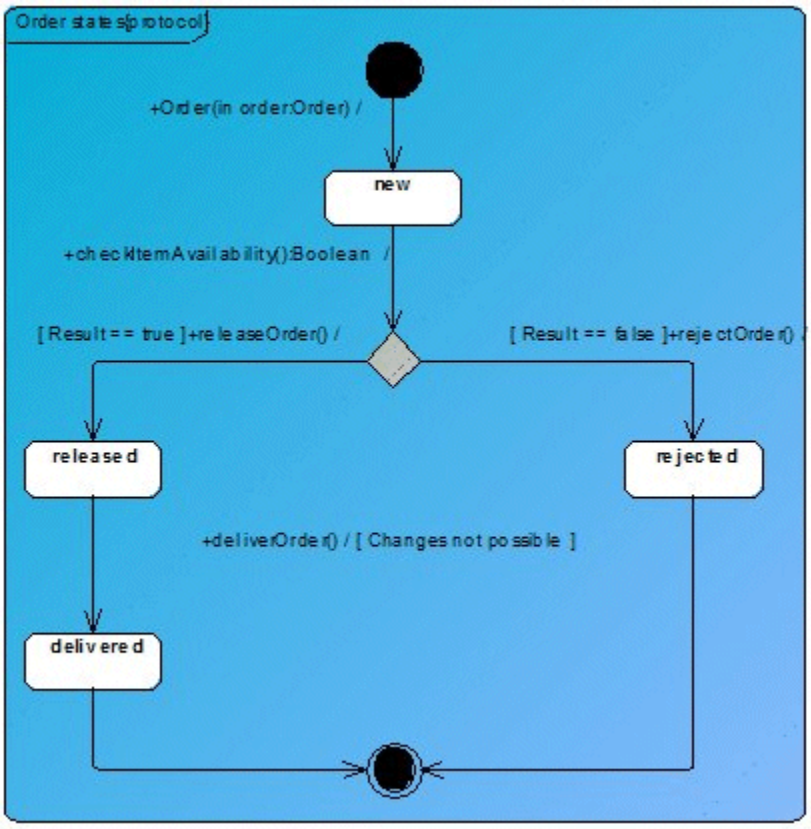


Figure 24: **Order states** Protocol state machine diagram

The Protocol state machine diagram in Figure 24 shows how invocations of these operations result in corresponding state changes. Instead of normal status transitions, protocol state transitions (UML type **ProtocolTransition**) are used in protocol state machines.

Protocol state transitions have a different notation than normal state transitions. The notation for a protocol state transition is: [Precondition] Event / [Postcondition]

The precondition is mapped to the **preCondition** property of the UML type **ProtocolTransition**, and the postcondition to the **postCondition** property. As for normal state transitions, the event is mapped to the **trigger** property, and the triggers are assigned events of the UML type **CallEvent**, which in turn refer to the corresponding operations for the class, which means that the corresponding operation is displayed as the event in the diagram.

2.3 Special features in ARIS UML Designer

2.3.1 Diagram content

The UML specification states that product manufacturers can extend the typical content it proposes for the different diagram types with content of other UML diagram types.

ARIS UML Designer supports any UML content for structure diagrams. In addition to the elements provided for the corresponding diagram type, you have the option of modeling all other UML constructs. This enables overview diagrams to be created, for example containing both class hierarchies and complete state machines and interactions in Sequence diagram notation. This also enables UML elements that are actually assigned to different diagram types to be linked to one another graphically.

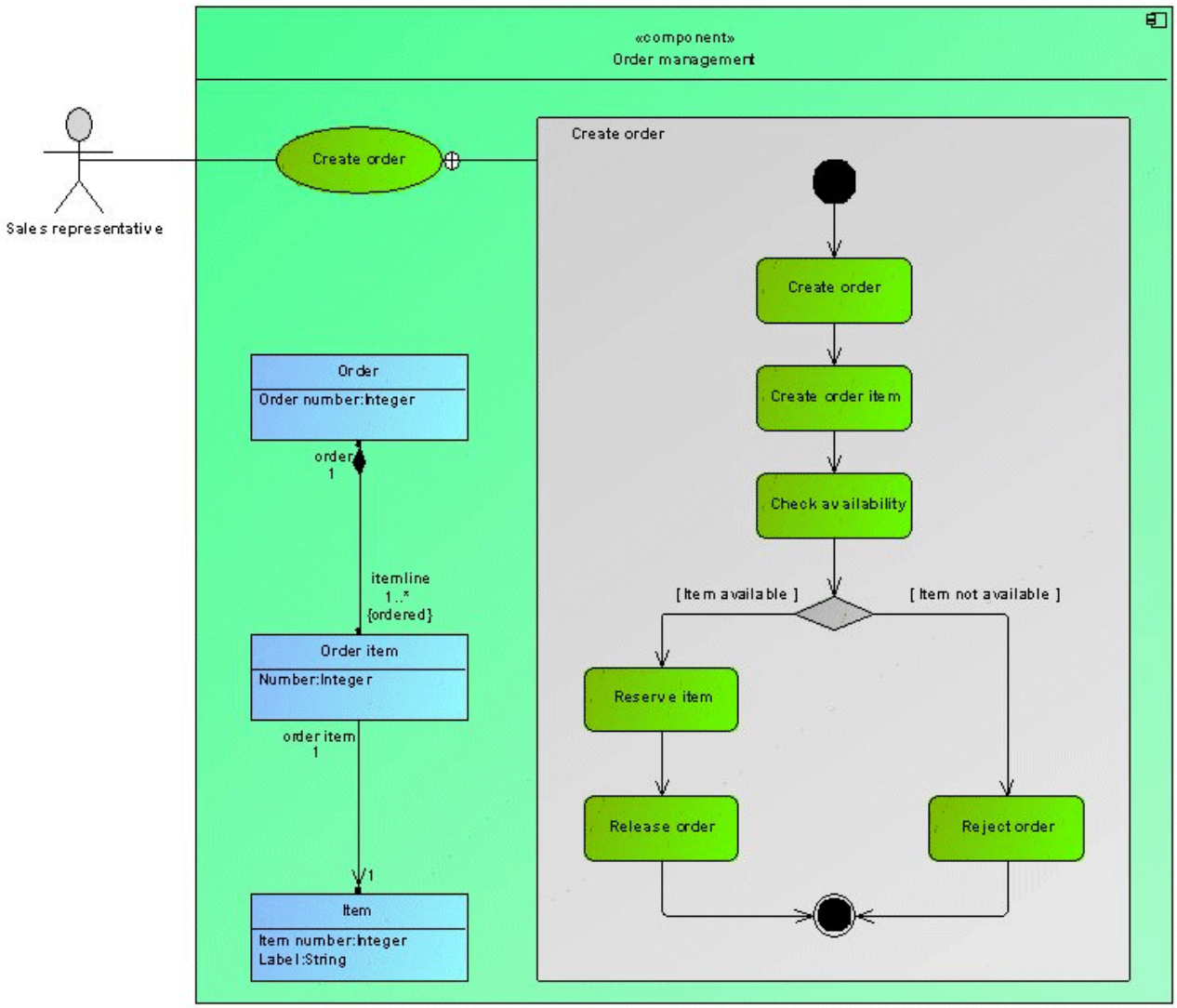


Figure 25: UML notation from different diagram types in a Class diagram

Figure 25 shows elements from Class, Use case, Component and Activity diagrams together in a single Class diagram.

2.3.2 Names of UML elements

The UML specification describes a small number of element types whose elements cannot have names. Examples of these types include Comment and Generalization. Because of the mapping of UML elements to ARIS objects, these types also have a name in ARIS UML Designer.

2.3.3 Multilingual capability

The UML specification does not include a facility for maintaining element names, comments, or other text properties in different languages. The only exception is the UML type **OpaqueExpression**, which allows the containing expression to have multiple values; you can specify a language for each of these values. However the term **Language** has a broader meaning than in ARIS and can also refer to a programming language.

ARIS UML Designer supports the familiar ARIS multilingual capability for UML content, but only for element and diagram names, comments, and descriptions. All other text properties are not multilingual.

2.4 The UML metamodel

A special feature of the UML specification is that it defines UML using a subset of UML. It uses Class diagrams to describe which UML element types exist, what properties they have, and how they are related to one another. The entirety of what is shown in these Class diagrams is referred to as the UML metamodel[10]. In this metamodel, the UML element types are defined in the form of metaclasses. The various properties of the UML element types are described using attributes and associations. Abstract metaclasses define properties that are shared by several different UML element types and and whose metaclasses inherit from these abstract metaclasses.

The architecture for description and implementation of metamodels is described in the MOF[11] standard published by the OMG. An ARIS-specific MOF implementation called OMF[12] represents a central component of the architecture of ARIS UML Designer.

Basic knowledge of the UML meta model is useful to understand UML and is essential for modeling UML profiles (see section 6**UML profiles**).

A short extract of the UML metamodel is introduced below, which defines those UML elements that are used in the Class diagram from Figure 1. The metamodel diagrams shown largely correspond to the relevant originals from the UML 2.5 specification. However, in some cases they have been adapted to the needs of this section by omitting individual metaclasses that are not dealt with further here, or by adding content from other diagrams in the UML specification.

---

[10] Apart from the Class diagrams, the UML specification also includes numerous OCL expressions, which describe consistency conditions on the one hand and the implementation of complex queries on the other. These are also part of the UML metamodel but will not be discussed further in this UML introduction.

[11] MOF stands for Meta Object Facility

[12] OMF stands for Object Modeling Framwork

The section headings used below correspond to the headings of the sections from the UML specification that contain the corresponding diagrams, so that you can easily find them in the UML specification to extend your knowledge of the subject.

2.4.1 Common Structure

2.4.1.1 Root

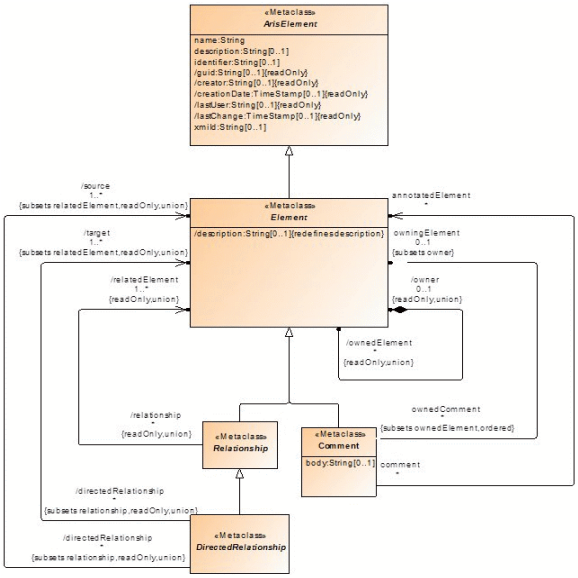


Figure 26: **Root** diagram from the UML specification with ARIS-specific extensions

Figure 26 shows the **Root** diagram from the UML specification, supplemented with ARIS-specific extensions. The metaclasses from the UML metamodel form an inheritance hierarchy with the abstract metaclass **Element** at the top. In this context, "abstract" means that there are no UML elements of the **Element** type, only that properties are defined here for other element types that inherit from Element. Abstract classes in UML diagrams are indicated by the name being written in italics.

In ARIS UML Designer, the UML metaclass **Element** inherits from an abstract metaclass **ArisElement**. This means that all UML elements are extended with ARIS-specific properties such as **creator**, **creationDate**, **name**, **description**, and **identifier**[13].

The non-abstract **Comment** metaclass inherits from the **Element** metaclass. Each UML element can have any number of these comments through its **ownedComment** property. In turn, a comment can refer to any number of UML elements through its **annotatedElement** property.

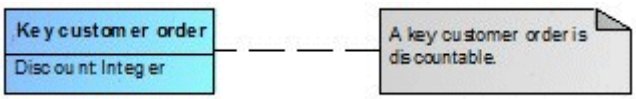


Figure 27: Comment with class as annotated element

[13] A slash **/** in front of an attribute name means that its value is derived from other properties at runtime. This also applies to ARIS properties such as **creator** or **guid**, as they are not mapped internally to ARIS attributes but result directly from the ARIS object.

Comments are displayed in diagrams as a rectangle with a turned-down corner. The text within the comment shows the value of the **Comment::body** property. The **Comment::annotatedElement** property is shown as a dotted edge to the annotated element in the diagram (see Figure 27).

The **Element::/description** property also represents an ARIS-specific extension of the UML metamodel. It redefines the ARIS property **ArisElement::description** for UML elements by deriving the description from the first comment that the Element has[14]. This means that UML elements have the ARIS property **description** in ARIS UML Designer. However, its value is not lost when exchanging data with other tools via XMI as it is saved as **ownedComment** for the element. Conversely, it means that every UML element that is imported into ARIS UML Designer via XMI and has a comment automatically has a value for its **description** property.

The Root diagram in Figure 26 also shows that in theory every UML element can have other UML elements. For this purpose, **Element** defines two properties **/owner** and **/ownedElement**. Both of these are so-called **derived unions**. This means that you can ask an element for its owner or the elements it contains, but other properties of the **Element** type and its specializations specify the way in which the element can contain other elements, or the way in which it can be contained in another element. An initial example of this is included in the same diagram. For the **Element::ownedComment** property there is **subsets ownedElement** and for the **Comment::owningElement** property there is **subsets owner**. This means that whenever an element is asked for the value of its **ownedElement** property, the values it returns include the value of the **ownedComment** property, i.e., the comments that it has. The same applies to asking a comment for its **owner** property. In this case the value of the **owningElement** is returned.

In addition, the Root diagram contains the two abstract metaclasses **Relationship** and **DirectedRelationship**. All metaclasses that primarily define semantics for relationships between elements inherit from these classes. These relationship elements are frequently shown as graphic edges in diagrams. This metamodel concept includes the two metaclasses **Generalization** and **Association** as examples of these relationship types.

[14] For this purpose, the **isOrdered** flag is set for the **Element::ownedComment** in ARIS UML Designer, while the UML metamodel does not originally provide ordering here.

2.4.1.2 Namespaces

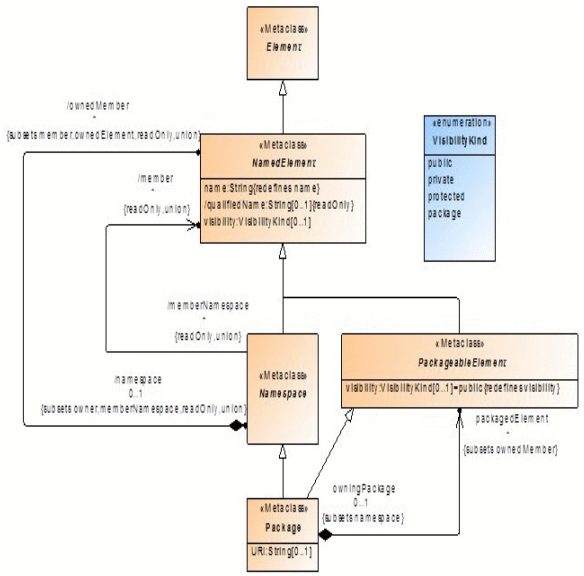


Figure 28: **Namespaces** diagram from the UML specification

Figure 28 shows the **Namespaces** diagram from the UML specification. It defines the three abstract types **NamedElement**, **Namespace**, and **PackageableElement**, and the non-abstract type **Package**. NamedElement defines the **name** property, which means that only UML elements whose type inherits from **NamedElement** can actually have a name. As all objects can have a name in ARIS and Element inherits from **ArisElement** in the ARIS implementation of the UML metamodel, the suffix **{redefines name}** for **NamedElement::name** specifies that a second **name** property is not defined, but that this definition of **name** replaces the **ArisElement::name** definition in the inheritance hierarchy for **NamedElement**. Both properties – **ArisElement::name** and **NamedElement::name** – are mapped to the ARIS attribute type **Name** in the ARIS object model.

Namespace is an abstract type for UML elements whose contained elements are differentiated using their name. The non-abstract type **Package**, which can contain elements of the abstract type **PackageableElement**, inherits from this.

The Namespaces diagram in Figure 28 also contains a further example of **derived unions**: The **Package::packagedElement** property contributes to the **Namespace::member** property, which in turn contributes to the **Element::ownedElement** property.

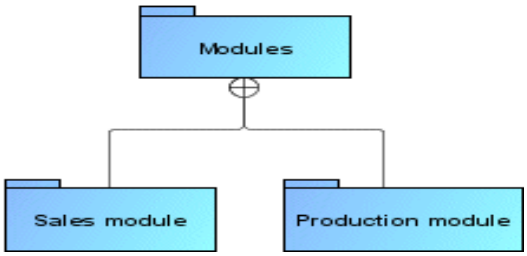


Figure 29: Example of **owningPackage** edges

The **Package::packagedElement** or **PackageableElement::owningPackage** property can be represented graphically as an edge in diagrams (see Figure 29).

2.4.1.3 Types and Multiplicities

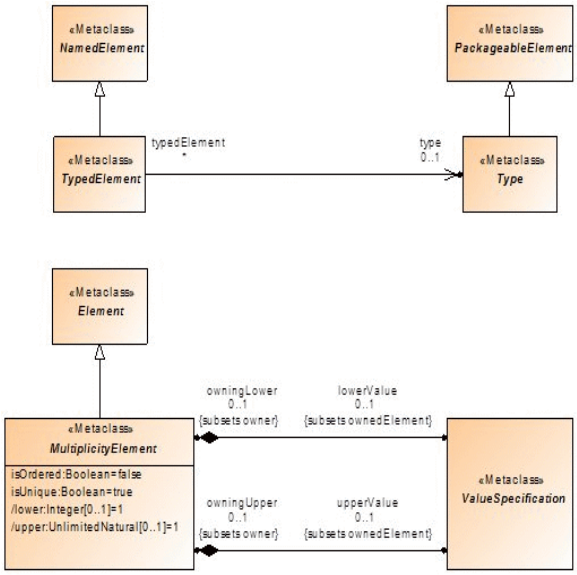


Figure 30: **Types and Multiplicities** diagram from the UML specification

Figure 30 shows the **Types and Multiplicities** diagram from the UML specification. It defines the abstract metaclasses **Type**, **TypedElement**, and **MultiplicityElement**.

Metaclasses whose elements can be assigned a type inherit from the abstract metaclass **TypedElement**. Typical examples of these elements are attributes (UML metaclass **Property**) and parameters (UML metaclass **Parameter**).

The abstract metaclass **MultiplicityElement** is the base class for all metaclasses whose elements can have a multiplicity. Once again, attributes and parameters are typical examples here. At this point, UML 2 is more complex than UML 1.x in the sense that the upper and lower limit for a multiplicity is no longer a primitive data type (not a simple number) but is a UML element whose metaclass inherits from the abstract type **ValueSpecification**. In addition to the corresponding **MultiplicityElement::lowerValue** and **MultiplicityElement::upperValue** properties, two further properties **MultiplicityElement::/lower** and **MultiplicityElement::/upper** are defined with a primitive data type, but these are derived from the assigned ValueSpecification elements.

2.4.2 Values

2.4.2.1 Literals

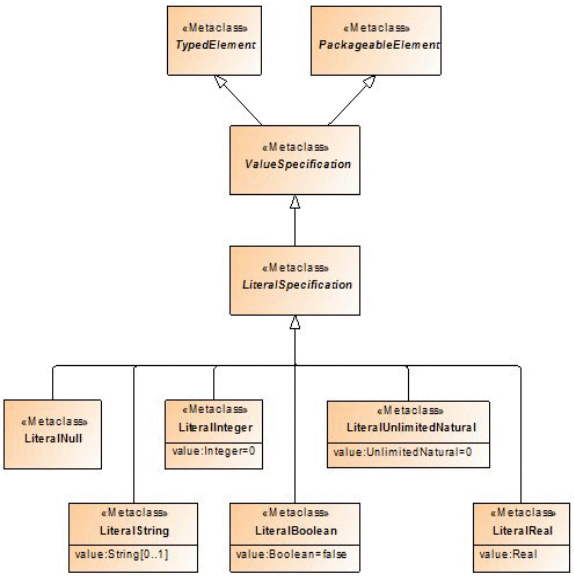


Figure 31: **Literals** diagram from the UML specification

Figure 31 shows the **Literals** diagram from the UML specification. It defines some specializations of the ValuesSpecification metaclass[15]. Typically, for the **MultiplicityElement::lowerValue** property you will use an element of the **LiteralInteger** type and for the **MultiplicityElement::upperValue** property an element of the **LiteralUnlimitedNatural** type.

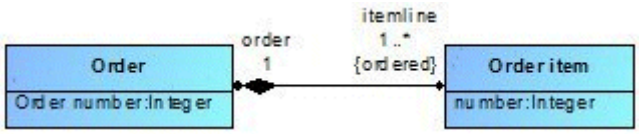


Figure 32: Class diagram with attributes, association ends, and multiplicities

[15] In addition to the specializations of ValueSpecification shown in the **Literals** diagram, there are others but these will not be discussed in more detail here.

The Class diagram in Figure 32 shows two examples of multiplicities:

The **order** association end **1**, which is a shortened form of 1..1, i.e., the **/lower** and **/upper** properties each have the value **1**. The specific elements of the **ValueSpecification** type on which the two values are based is not shown by the graphic notation in the diagram.

The **itemline** association end has a multiplicity of **1..***, i.e., the **lower** property has the value **1** and the **upper** property has the value *, where * stands for unlimited.

2.4.3 Classification

2.4.3.1 Classifiers

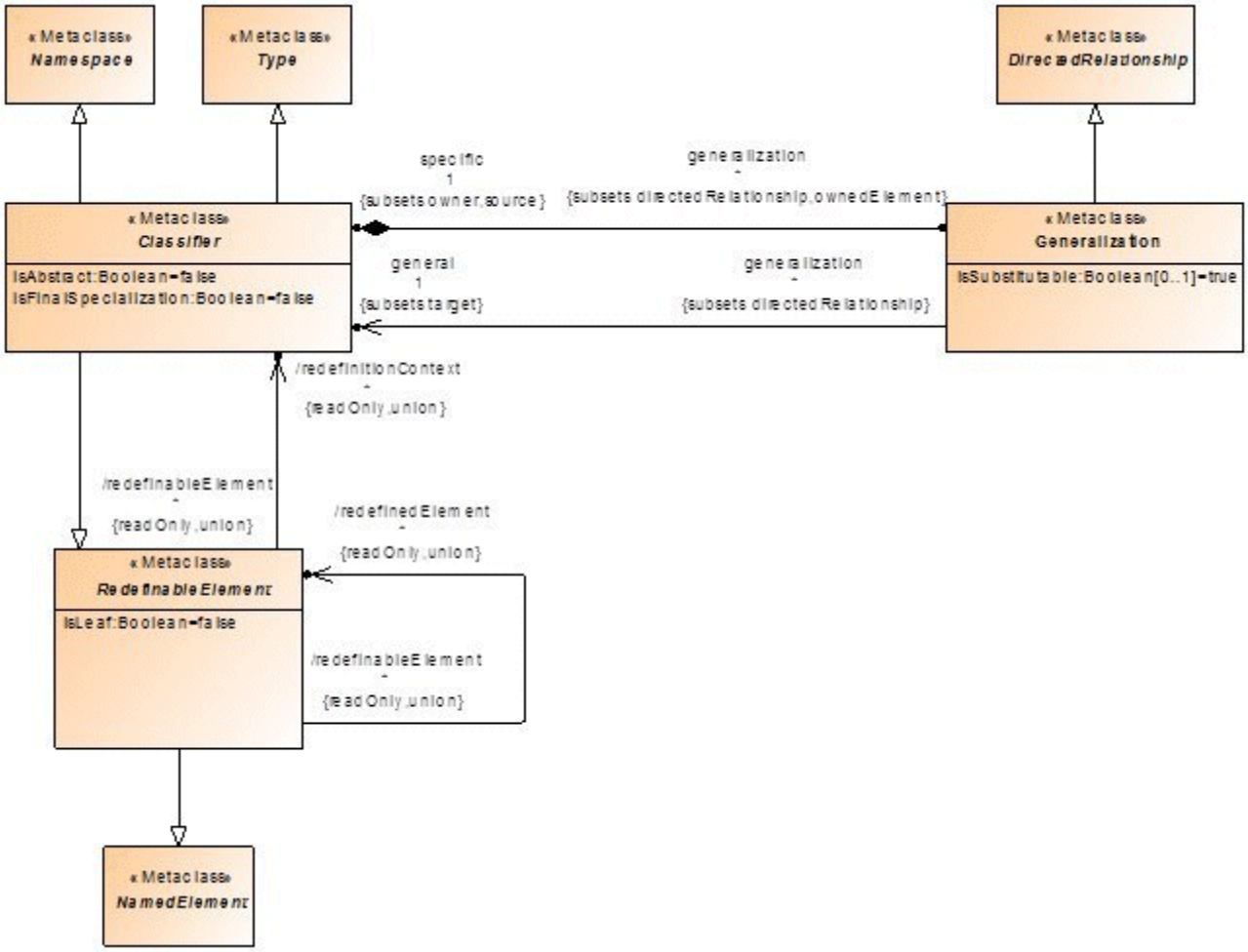


Figure 33: **Classifiers** diagram from the UML specification

Figure 33 shows the **Classifiers** diagram from the UML specification. It defines the two abstract metaclasses **RedefinableElement** and **Classifier**, and the non-abstract metaclass **Generalization**.

RedefinableElement is the basis for all UML element types for which it will be possible for a UML element to replace another existing UML element in a particular context. The UML specification itself uses this feature in its own diagrams. Figure 28 shows an example of this. The **PackageableElement::visibility** attribute replaces the corresponding attribute from the **NamedElement** base class, as unlike **NamedElement::visibility** it has a default value.

Classifier is the basis for all UML element types for whose elements inheritance relationships can exist. These inheritance relationships are based on the **Generalization** metaclass.

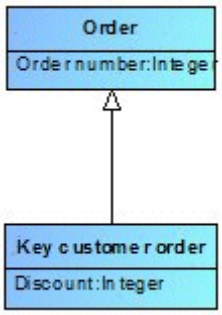


Figure 34: Generalization relationship between two classes

Figure 34 shows this kind of generalization relationship between the two classes **Key customer order** and **Order**. This relationship itself is a UML element of the **Generalization** type. In line with the compositional property **Classifier::general** this element is part of the derived class **Key customer order** and its **Generalization::general** property refers to the base class **Order**.

2.4.3.2 Features

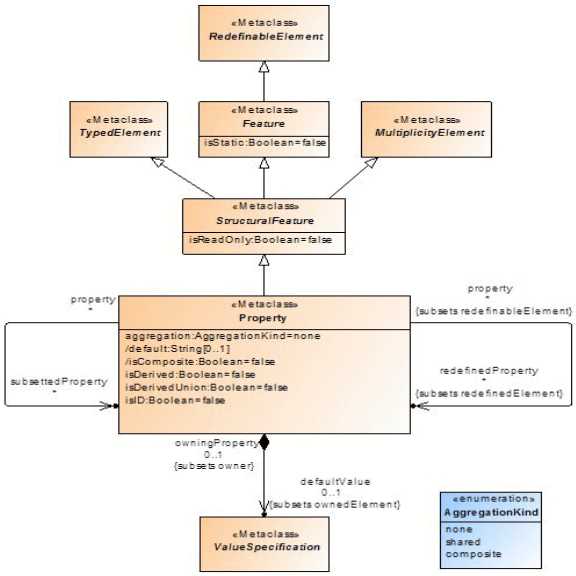


Figure 35: **Features** diagram from the UML specification

Figure 35 shows an extract from the **Features** diagram, supplemented with content from the **Properties** diagram in the UML specification. At the center is the non-abstract metaclass **Property** with its base classes. Attributes of classes and association ends are based on the **Property** metaclass.

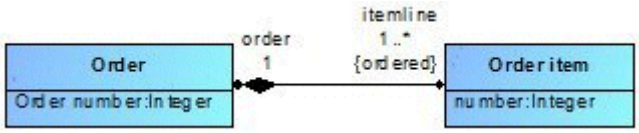


Figure 36Class diagram with attributes and association ends

The Class diagram in Figure 36 shows four UML elements of the **Property** type: the **Order number** and **Number** attributes, and the **order** and **itemline**[16] association ends. The association edge shows a shaded black diamond at the opposite end of **order**. This means that the order items are part of the order and are existentially dependent on it, i.e., deleting the order includes deleting its order items. This kind of association is known as a composition. At the **order** association end, the shaded diamond is displayed when the **Property::aggregation** property for the **itemline** association end has the value **AggregationKind::composite**.

[16] These diagrams actually also include the two **Attribute** association ends for the respective opposite class. This is dealt with in more detail in the section on the **Association** metaclass.

2.4.4 Structured Classifiers

2.4.4.1 Classes

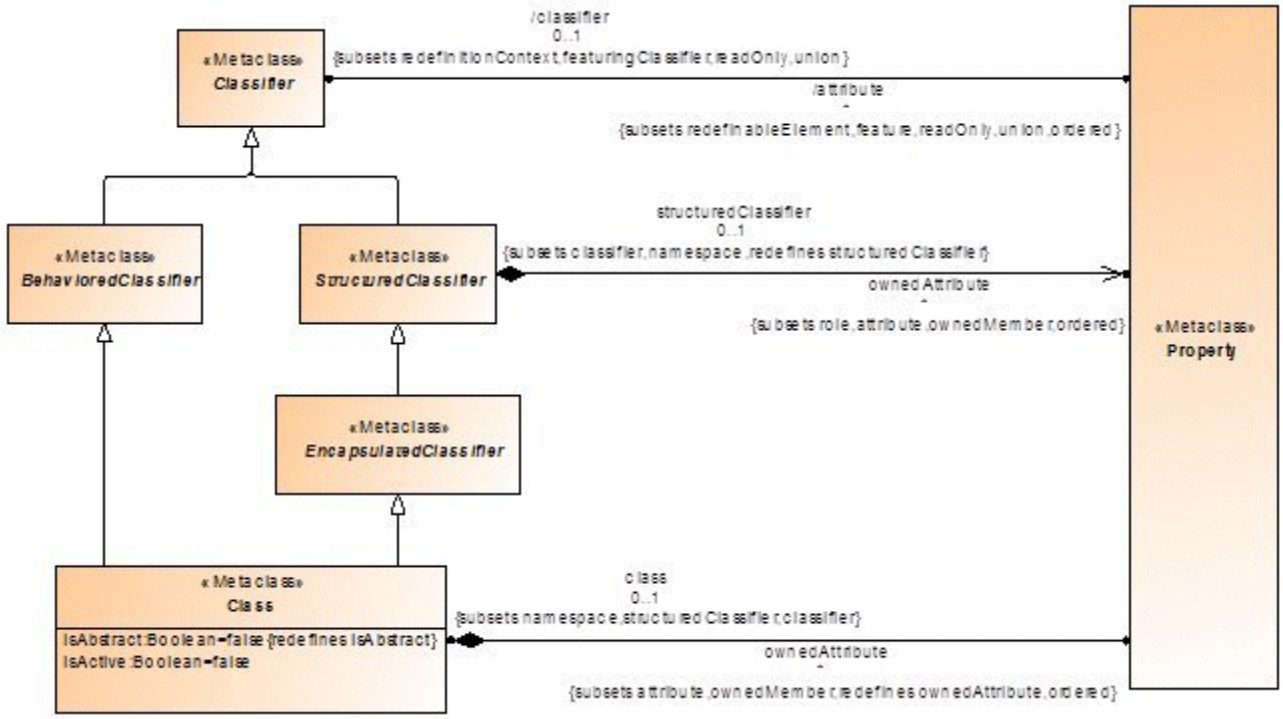


Figure 37: **Classes** diagram from the UML specification

Figure 37 shows an extract from the **Classes** diagram supplemented with content from the **Structured Classifiers** and **Encapsulated Classifiers** diagrams in the UML specification. The central metaclass in this diagram is Class. The three abstract base classes **BehavioredClassifier**, **StructuredClassifier**, and **EncapsulatedClassifier** are only included here to illustrate the inheritance relationship between Class and Classifier. They define additional properties that will not be discussed further in this introduction.

The diagram also shows that attributes (UML type **Property**) can theoretically be assigned to all classifiers using the derived property **Classifier::/attribute**. The type of the specific assignment is defined by the corresponding specializations. For the **Class** metaclass it is the compositional property **ownedAttribute**. An example of the representation of attributes in classes is shown in Figure 36.

2.4.4.2 Associations

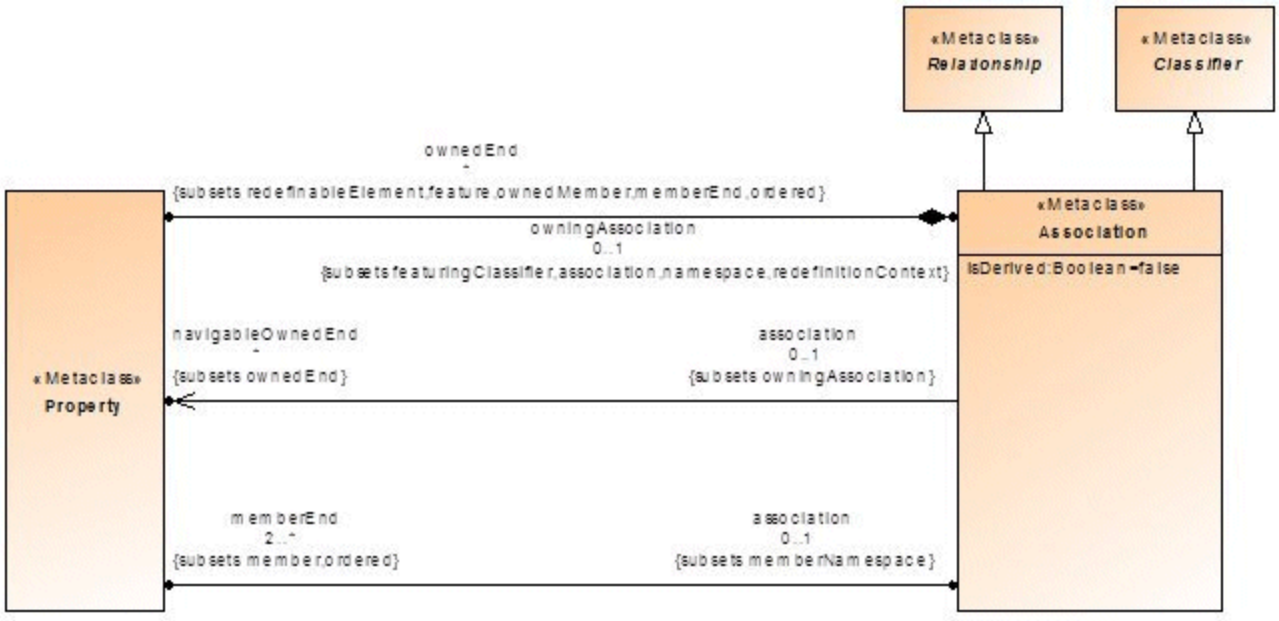


Figure 38: **Associations** diagram from the UML specification

Figure 38 shows an extract from the **Associations** diagram from the UML specification. It defines the **Association** relationship type as the central metaclass. Associations are used to model structures made up of one or more classes or other UML types. The association is not directly linked to the partners in the relationship, but uses an element of the UML type **Property**, often referred to as association ends in this context. The **Association::memberEnd** property is critical here, and its multiplicity limit of 2 means that an association must have at least two association ends[17].

These association ends can either be contained in the association itself using the **Association::ownedEnd** property, or can be an attribute of the opposite relationship partner (in the case of a class this would be the **Class::ownedAttribute** property). If the association end is an attribute of the opposite relationship partner, the association is navigable towards the association end. Alternatively, an association end that is not an attribute (i.e., it belongs to its association as ownedEnd) can also be navigable if it is also assigned to the association as navigableOwnedEnd.

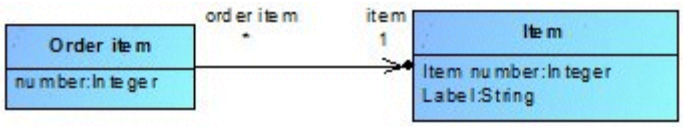


Figure 39: Class diagram with an association

[17] Associations with two ends are also referred to as binary associations. These are normally shown as a graphic edge in diagrams. If an association has more than two ends, we refer to a multiple association. Multiple associations are shown in diagrams as diamonds with edges to the relationship partners.

The association in Figure 39 links two classes with one another. The **item** association end is an attribute of the **Order item** class (**Class::ownedAttribute** property). You can identify this by the shaded black dot at the end of the edge. As an attribute of the class, the end of a binary association is automatically navigable and is shown with an arrow head[18]. This means that an order item recognizes the item assigned to it. By contrast, the **order item** association end is part of the association (**Association::ownedEnd** property) and is not navigable.

[18] If both ends of a binary association are navigable, displaying the arrow heads in the diagram is optional.

2.4.5 Simple Classifiers

2.4.5.1 DataTypes

Figure 40: **DataTypes** diagram from the UML specification

Figure 40 shows an extract from the DataTypes diagram from the UML specification. It defines the two non-abstract metaclasses DataType and PrimitiveType. These data types differ from classes in the sense that there is no object identity for their instances. Instances of data types are only differentiated from one another based on their value. Simple data types (PrimitiveTypes metaclass) also have an internal structure.

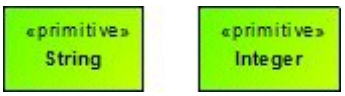


Figure 41: Class diagram with two primitive data types

3 ARIS UML Designer overview

This section provides a brief introduction to the individual components of ARIS UML Designer and their key functionalities.

ARIS UML Designer provides different functional components depending on the selected working environment in the perspective.

3.1 Specifying the working environment

You can specify the perspective and therefore the working environment by selecting the **Select perspective** menu item.

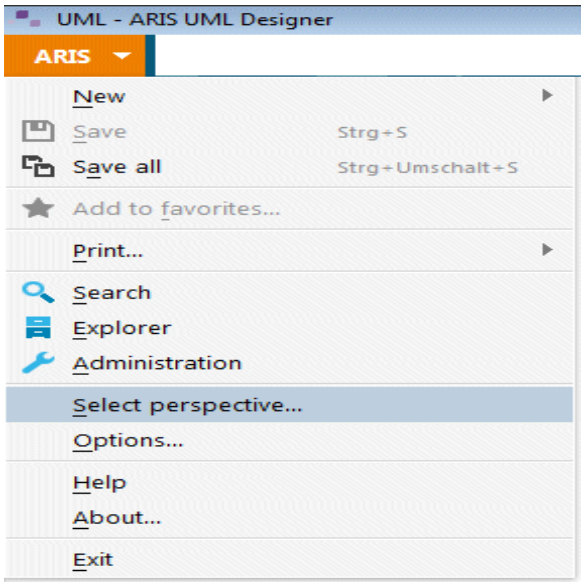


Figure 42: Menu item for specifying the perspective

This launches a wizard for specifying the perspective.

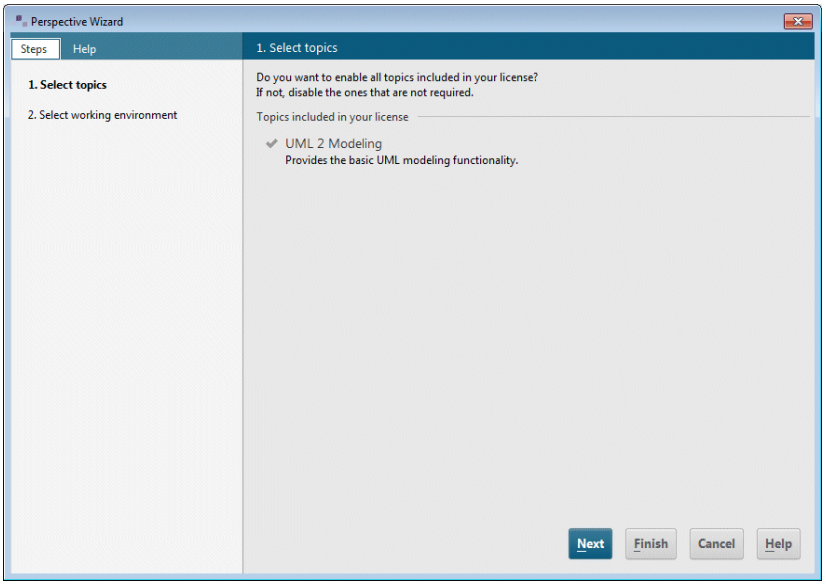


Figure 43: Topic selection in the Perspective Wizard

In contrast to ARIS Architect and ARIS Designer, apart from UML modeling no other license-dependent topics are available for ARIS UML Designer.

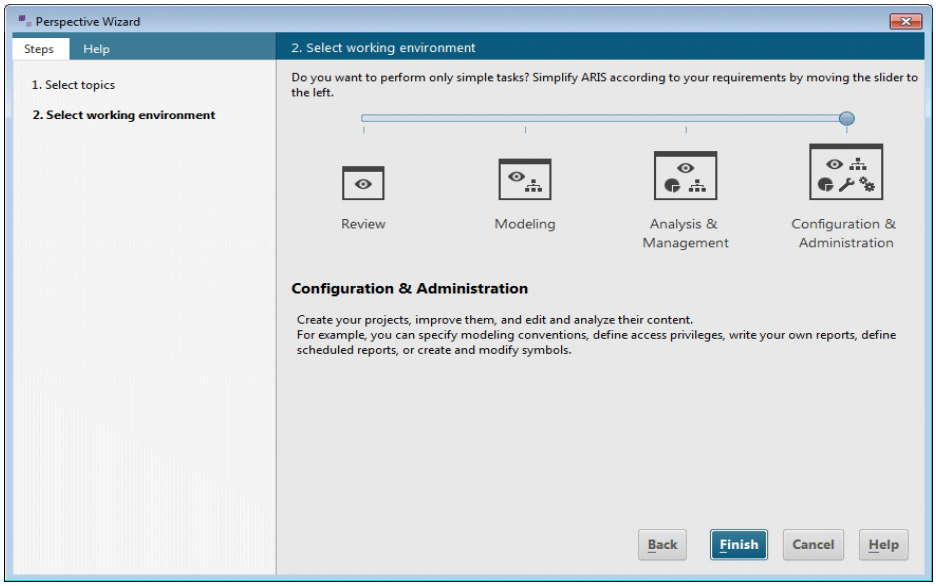


Figure 44: Working environments in the Perspective Wizard

By selecting the working environment, you adjust the range of functions in ARIS UML Designer to the work you are involved in.

Review allows read-only access to the UML content. It is not possible to make any changes to the data and diagrams.

Modeling allows you to edit the content.

Analysis & Management enables additional functionalities such as XML export and XML import.

Configuration & Administration includes administrative activities in ARIS such as editing method filters or configuration of reuse options for business process objects in UML, and creation and editing of scripts.

3.2 Explorer

When you launch ARIS UML Designer, the **Explorer** tab is displayed.

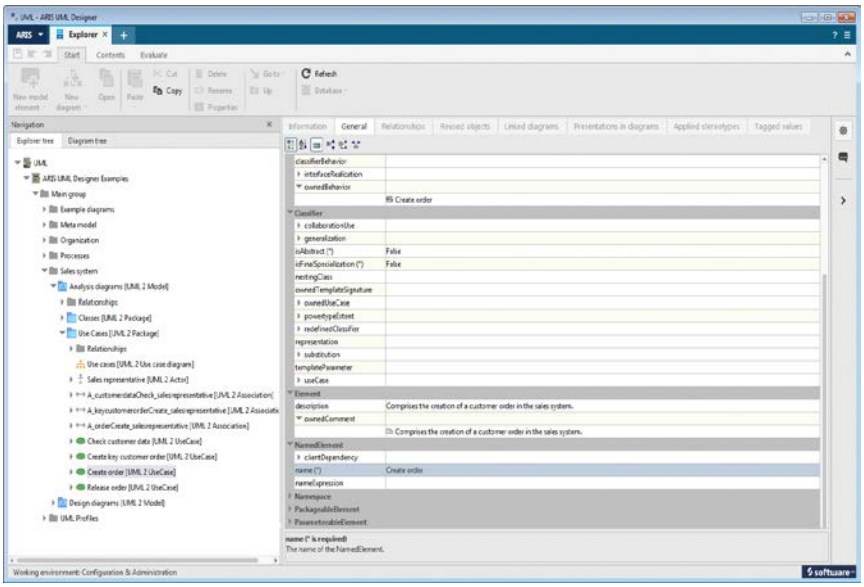


Figure 45: **Explorer** tab

Figure 45 shows the **Explorer** tab in ARIS UML Designer. The **Navigation** bar on the left-hand side contains two trees – the Explorer tree and the diagram tree. The properties of the element or diagram selected in the tree are displayed on the right-hand side. Alternatively, you can display the properties in a separate dialog by clicking **Properties** in the pop-up menu for an element or diagram.

The **Navigation** bar can be hidden using the  button at the right-hand edge of the window to create more space for the properties pages.

You can use the  button to show and hide the **Implicit changes** bar at the right-hand edge of the window. This area logs when a change to a UML element results in implicit changes to other UML elements. The **Implicit changes bar** is outlined in more detail in section 0.

You can use the  button to hide all areas except for the properties pages. Clicking the button again reverts to the previous state.

If you have closed the **Explorer** tab, you can re-open it by selecting **Explorer** in the ARIS menu.

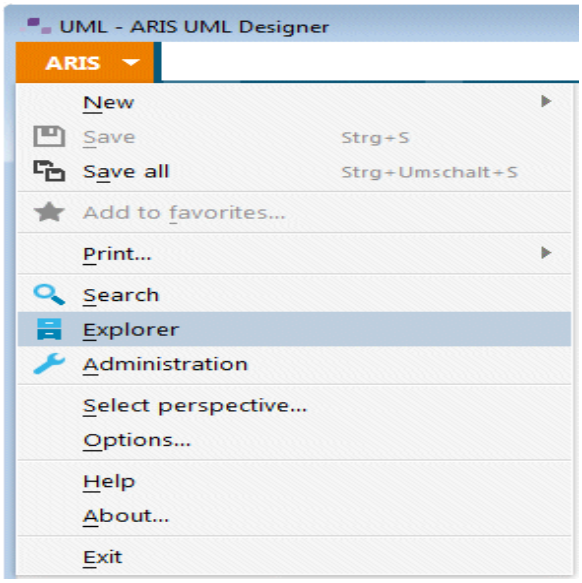


Figure 46: Menu item for displaying the **Explorer** tab

3.2.1 Navigation bar

3.2.1.1 Explorer tree

The Explorer tree shows the familiar ARIS group hierarchy with diagrams and elements. The tree includes both standard ARIS and UML content.

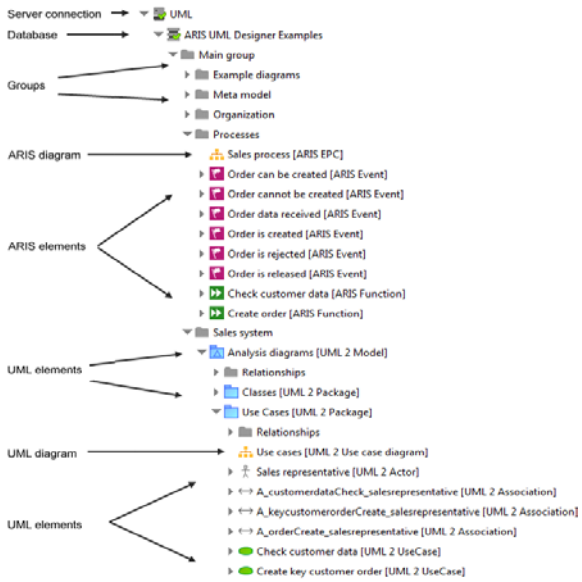


Figure 47: Groups, diagrams, and elements in the Explorer tree

A fundamental difference between ARIS and UML is that, unlike ARIS items, UML elements can form hierarchies, i.e., a UML element can contain other UML elements, and this is visible in the Explorer tree. The root of this kind of hierarchy of UML elements is always a UML element of the **Package**, **Model**, or **Profile** type. Only these three types can be directly contained in a group.

Essentially, only new groups, UML elements, and UML diagrams can be created in ARIS UML Designer. ARIS items and ARIS diagrams are displayed in the Explorer tree in ARIS UML Designer, but they cannot be created there.

3.2.1.2 Diagram tree

The diagram tree provides a view of the database grouped by diagram types. Particularly with small or medium-sized databases, it offers fast and uncomplicated access to diagrams.

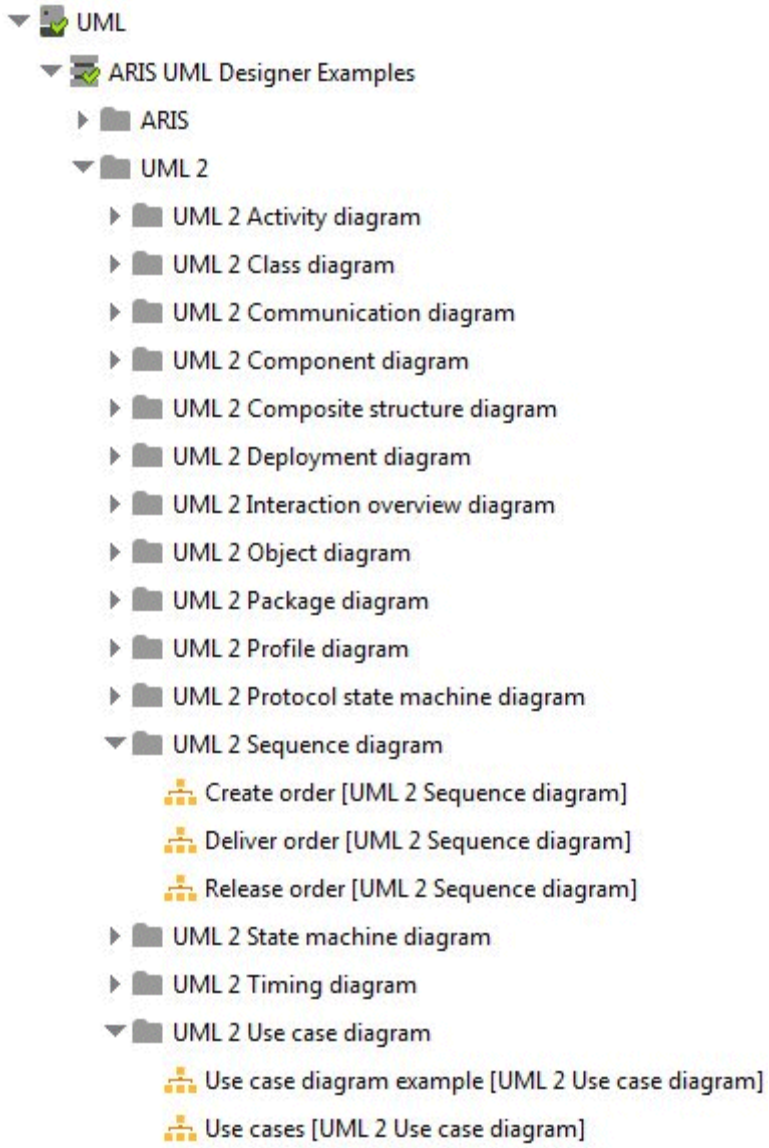


Figure 48: Diagram tree

Figure 48 shows the **Explorer** tab with the diagram tree in the **Navigation** bar. Below the database node are the two metamodel nodes **ARIS** and **UML**. These each contain the corresponding diagrams grouped by diagram type.

3.2.2 Properties pages

The properties of the element or diagram selected in the **Navigation** bar are displayed on several properties pages on the right-hand side of the **Explorer** tab. The most important properties pages are outlined below. Special properties pages relating to UML profiles or links between business process and UML modeling are explained in the corresponding sections of this document.

Essentially, all properties pages on the **Explorer** tab are also displayed in the Properties dialog for the element or diagram.

3.2.2.1 Information (elements, diagrams, groups)

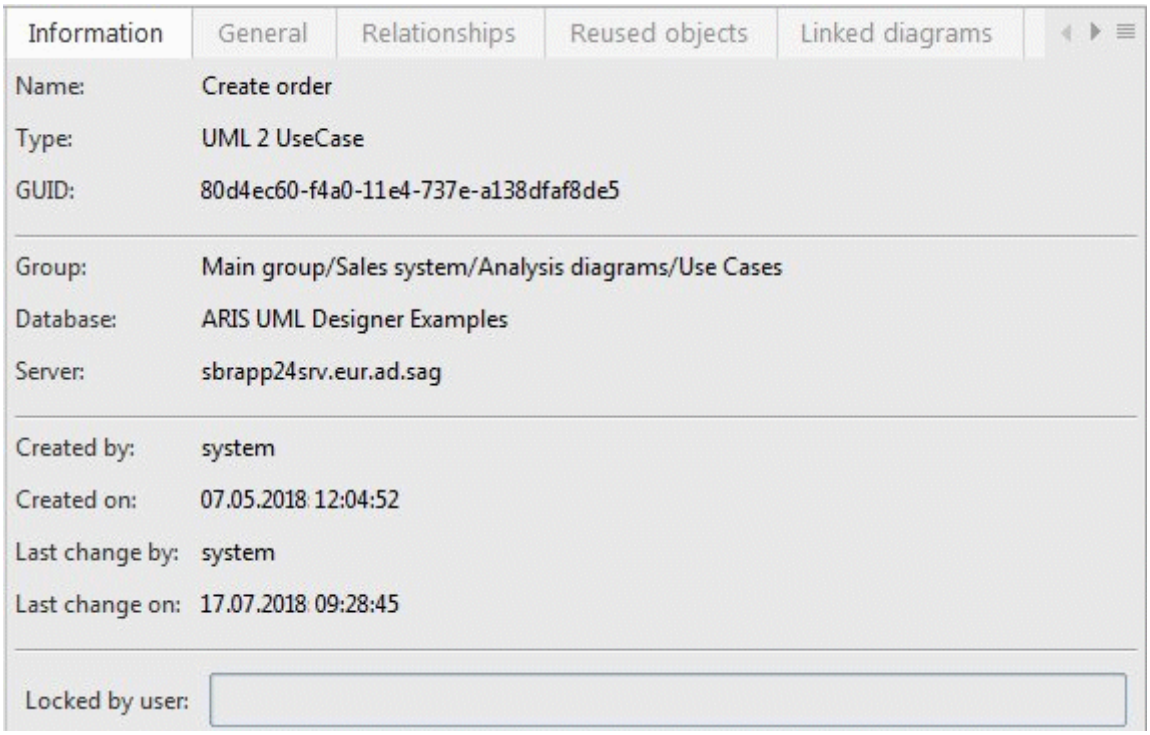


Figure 49: **Information** properties page

The **Information** properties page is displayed for groups, elements, and diagrams. It shows general and, in some cases, ARIS-specific properties of the selected element or diagram.

The **Group** property contains the complete path to the selected element or diagram, not only ARIS groups but also superior UML elements.

3.2.2.2 General (elements, diagrams, groups)

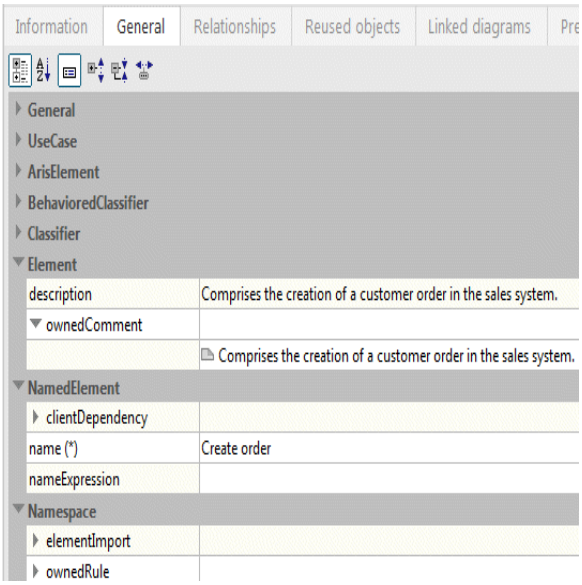


Figure 50: **General** properties page

The **General** properties page shows all properties that are specified in the metamodel as attributes of the metaclass for the selected element and as attributes of the meta diagram for the selected diagram. The lower section of the page shows a description of the selected property. Properties that the metamodel stipulates must have a value are indicated by an asterisk (*).

The **Show/Hide description area** button is used to set whether or not the description area is displayed.

The properties are grouped by the metaclasses and meta diagrams to which they are assigned as attributes in the metamodel. Alternatively, you can sort the properties alphabetically without displaying their metaclasses or meta diagrams (see Figure 51).

The **Categorized** and **Alphabetically** buttons are used to toggle between the properties being grouped by metaclasses and displayed alphabetically.

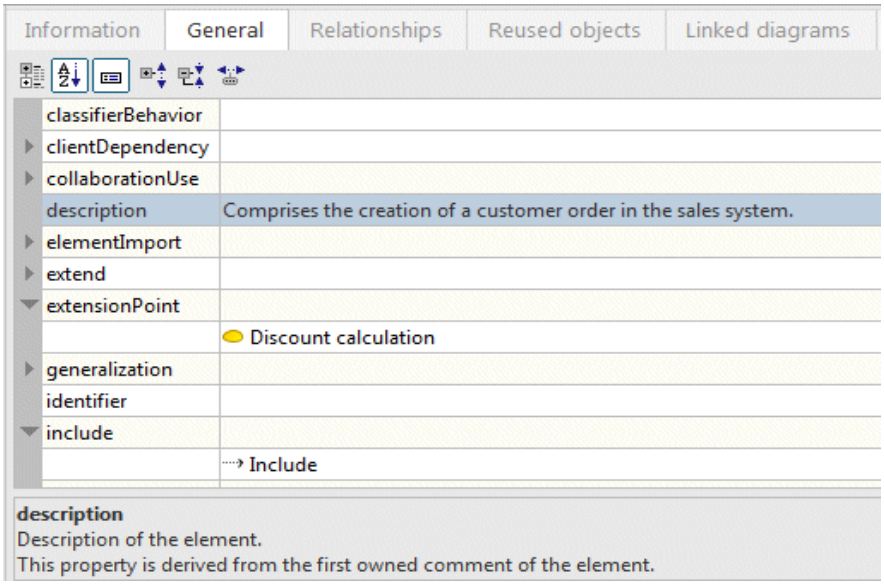


Figure 51: Alphabetical property display

Apart from a few exceptions[19] no properties that are based on derived meta attributes are displayed by default, i.e., their value has to be calculated at runtime based on other properties and elements. You have the option of also displaying derived properties.

The  **Show derived properties** button enables or disables the display of derived properties.

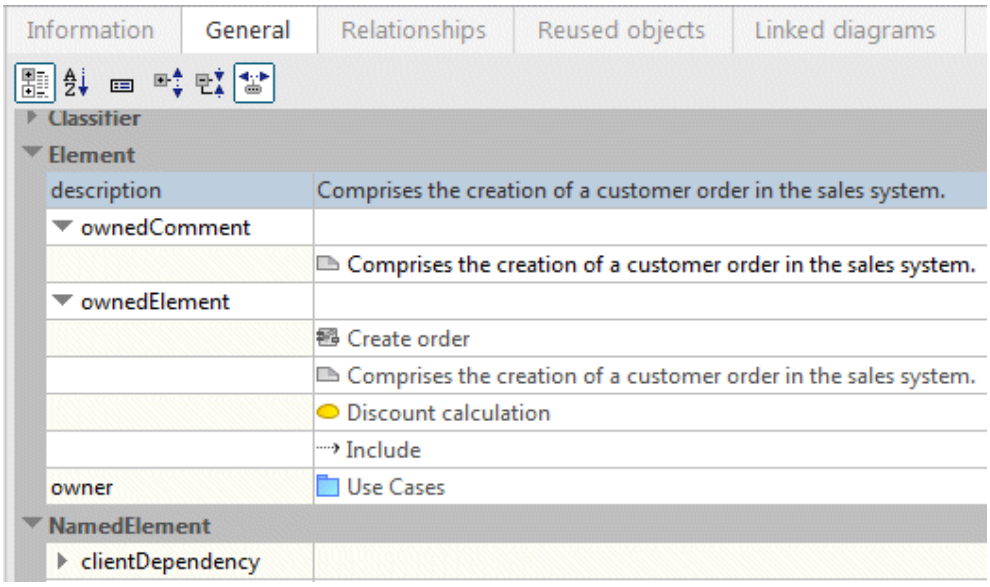


Figure 52: **General** properties page with display of derived properties

The example in Figure 52 shows that when displaying derived properties for the **Element** metaclass, i.e., for all UML elements, the **/ownedElement** and **/owner** properties are also listed (see Figure 26: **Root** diagram from the UML specification with ARIS-specific extensions). Their values are written in gray rather than black to indicate that these properties cannot be changed.

[19] These exceptions include Element::/description, MultiplicityElement::/lower, and MultiplicityElement::/upper.

Clicking the value of a property allows you to edit it. The type of editing permitted depends on the property type.

Properties of the **String** type can be edited directly in the text line:

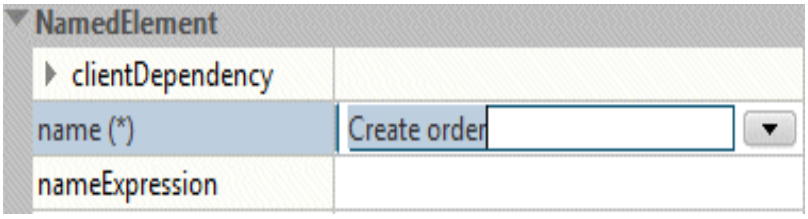


Figure 53: Text entry

Clicking the button on the far right opens a special text editing dialog:

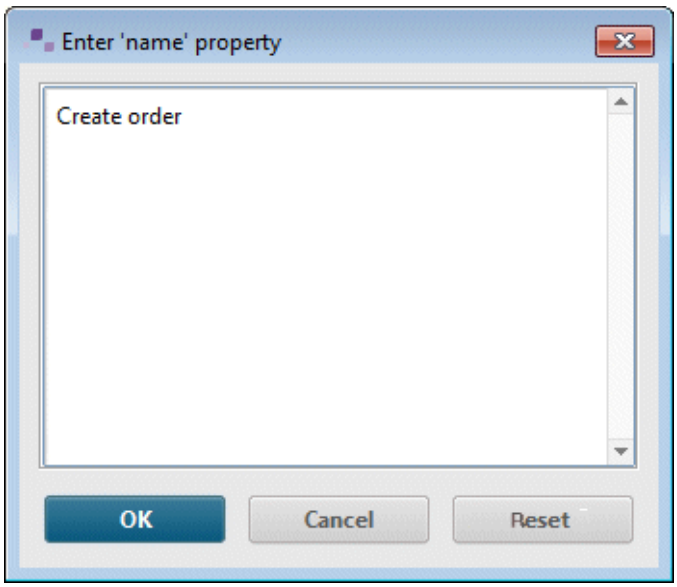


Figure 54: Editor for non-formattable text

If the property supports formatted text, this dialog provides the corresponding formatting tools. ARIS UML Designer only supports this for descriptions and comments:

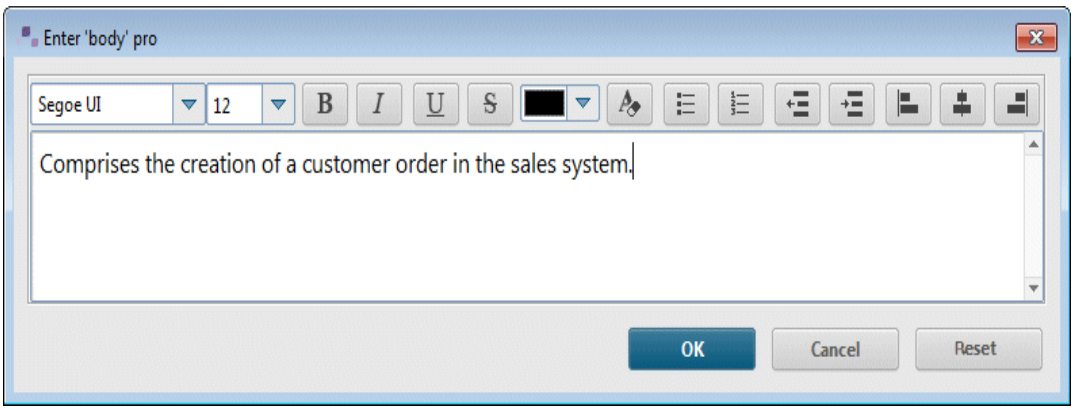


Figure 55: Editor for formattable text

If the value of the property is a UML element, direct editing in the text line is also possible:

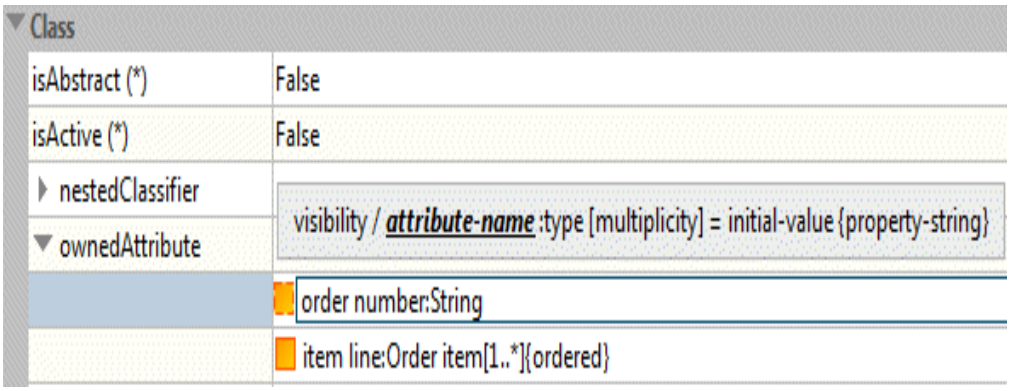


Figure 56: Text editing for a UML element in the general properties

During editing, a corresponding UML syntax help is shown above the text line. This UML-specific text editing option is available in ARIS UML Designer wherever the element is displayed in this text form – in the Explorer tree, in the properties pages for the superior element, and in diagrams. It is described in more detail in the Creating new elements in Explorer section.

The button on the far right is used to open a pop-up menu, which provides additional functionalities for the element:

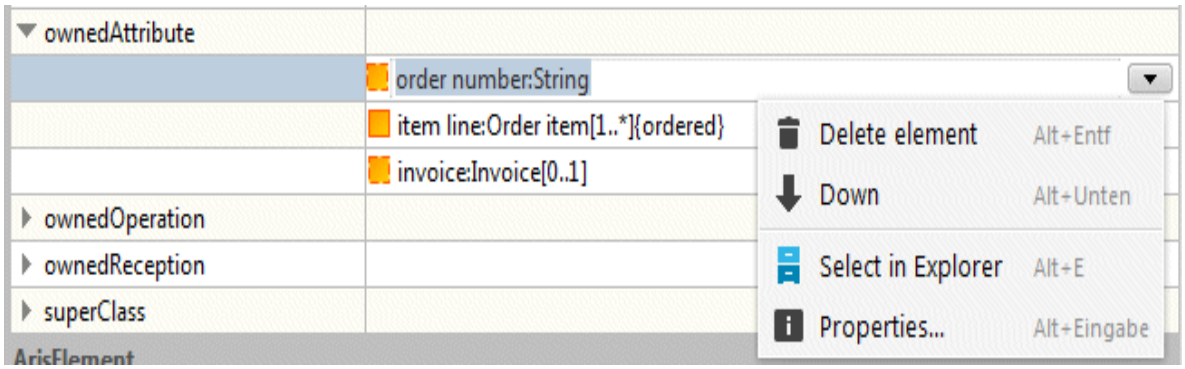


Figure 57: Additional functionalities for a UML element in the general properties

A pop-up menu is also available in the row of the table containing the name of the property. It is used to create corresponding new UML elements or to add existing elements to the property. The button for expanding the menu is available as soon as you click in the corresponding field in the table.

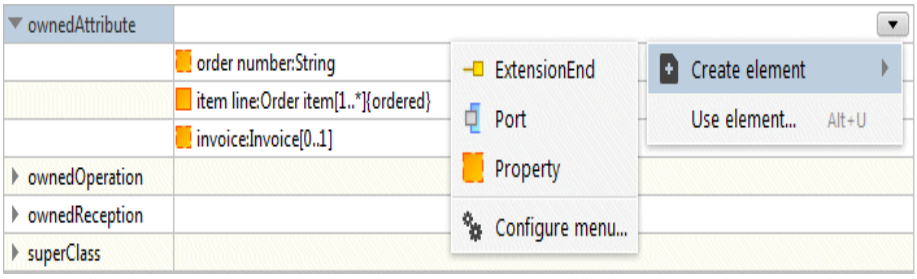
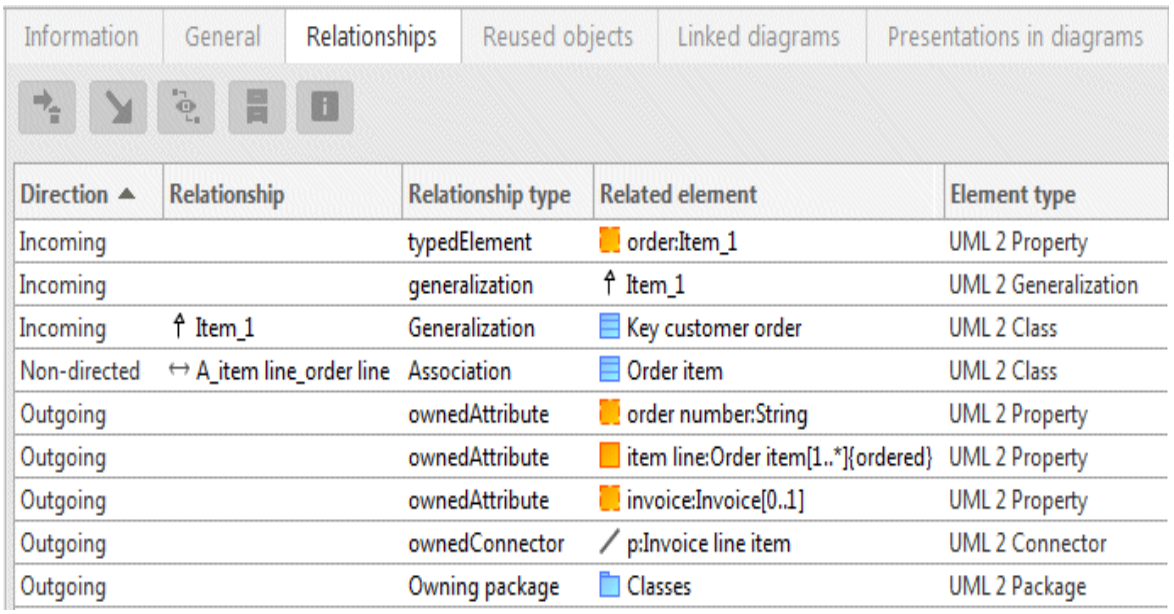


Figure 58: Creating new UML elements in the general properties

3.2.2.3 Relationships (elements)



Direction	Relationship	Relationship type	Related element	Element type
Incoming		typedElement	order:Item_1	UML 2 Property
Incoming		generalization	Item_1	UML 2 Generalization
Incoming	Item_1	Generalization	Key customer order	UML 2 Class
Non-directed	A_item line_order line	Association	Order item	UML 2 Class
Outgoing		ownedAttribute	order number:String	UML 2 Property
Outgoing		ownedAttribute	item line:Order item[1..*]{ordered}	UML 2 Property
Outgoing		ownedAttribute	invoice:Invoice[0..1]	UML 2 Property
Outgoing		ownedConnector	p:Invoice line item	UML 2 Connector
Outgoing		Owning package	Classes	UML 2 Package

Figure 59: **Relationships** properties page

The **Relationships** properties page displays all of the selected element's relationships with other elements. Alongside the direct relationships, those that appear as a direct graphic link between two elements in the diagram but actually represent a chain of elements and relationships are also displayed.

The binary association is an example of this kind of relationship. It links two classifiers using a graphic edge in the diagram, for example a user case and an actor. However, this association edge does not visualize a direct relationship but a chain of elements consisting of two association ends (UML type **Property**) and an association.

The **Relationships** properties page for the use case shows both the direct relationship between the use case and the association end as an incoming relationship of the **typedElement** type, and the indirect relationship with the actor, which is of the **Association** type.

You can call up the following functionalities for every relationship:

 Remove relationship

 Go to occurrence of linked element in Explorer

 Show element properties for linked element

The other two functionalities are only available in the Designer component and are described in the corresponding section.

3.2.2.4 Linked diagrams (elements)

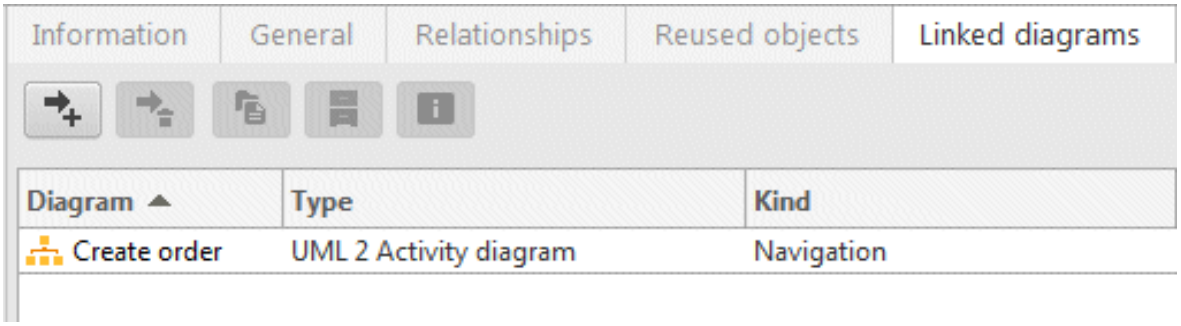


Figure 60: **Linked diagrams** properties page

All diagrams that are linked to the selected element are displayed here. If the selected element has a presentation in a diagram, it is displayed with an assignment symbol. This supports navigation to the linked diagrams in that diagram.

This kind of link can take three forms:

Ownership

The diagram belongs to the selected element. It appears as a child node of the element in the Explorer tree.

Navigation

The diagram has been assigned to the element for the purpose of navigation. This kind of link has no semantics. It is only used to provide a simple way of navigating from an element to a diagram. Unlike diagram assignments in the ARIS standard, no restrictions exist here. Every diagram type can be linked to every element type.

Implicit ownership

In this case, the diagram does not directly belong to the selected element. It is a behavior diagram[20] whose owner belongs to the selected element. For example, if you want to model the internal process of a use case in an Activity diagram, you create an activity as ownedBehavior of the use case and a corresponding Activity diagram for the activity. This Activity diagram is automatically linked to the use case through implicit ownership.

[20] Implicit ownership is not supported for structure diagrams. For example, if a child package of a package has Class diagrams, they are not linked to the package.

You can call up the following functionalities:

 Assign diagram

 Removed assigned diagram (only for assignments of Navigation type)

 Open diagram

 Go to occurrence of diagram in Explorer

 Show diagram properties

3.2.2.5 Presentations in diagrams (element)

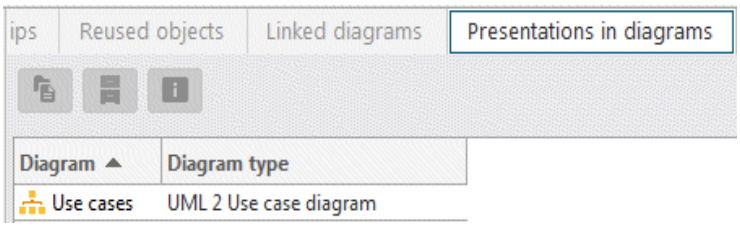


Figure 61: **Presentations in diagrams** properties page

The **Presentations in diagrams** properties page lists all diagrams that contain the selected element.

You can call up the following functionalities for each diagram:

Open diagram

Go to occurrence of diagram in Explorer

Show diagram properties

3.2.2.6 Presentations (diagrams)

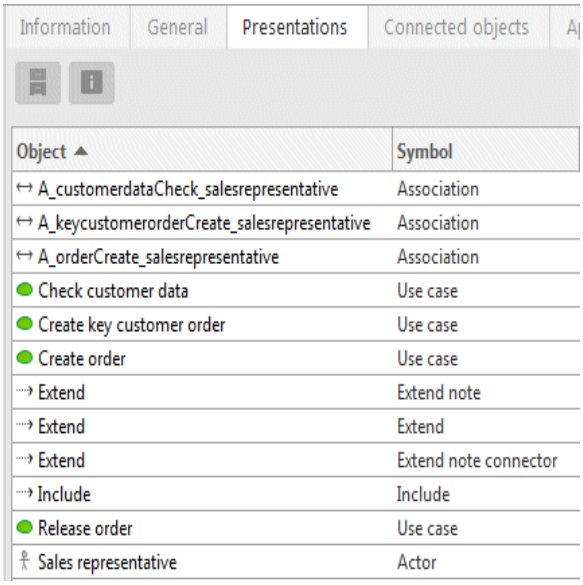


Figure 62: **Presentations** properties page

The **Presentations** properties page lists all elements that appear in the selected diagram.

You can call up the following functionalities for each element:

Go to occurrence of element in Explorer

Show element properties

3.2.2.7 Connected objects (diagrams)

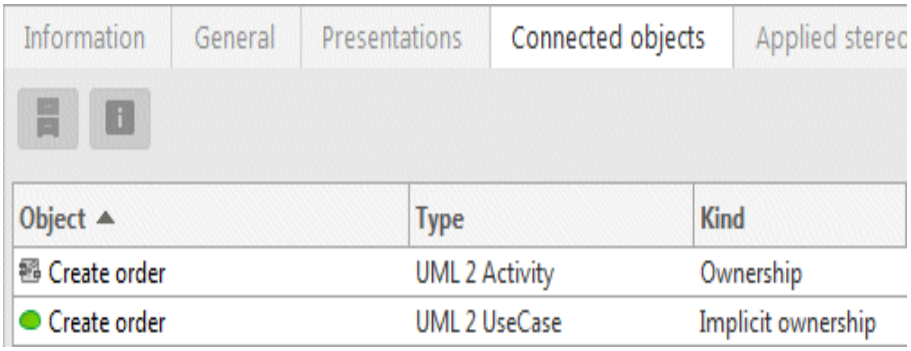


Figure 63: **Connected objects** properties page

The **Connected objects** properties page for a diagram shows the element that owns the diagram and optionally also the implicit owner in the case of a behavior diagram.

You can call up the following functionalities for each element:

 Go to occurrence of element in Explorer

 Show element properties

3.2.3 Properties dialog

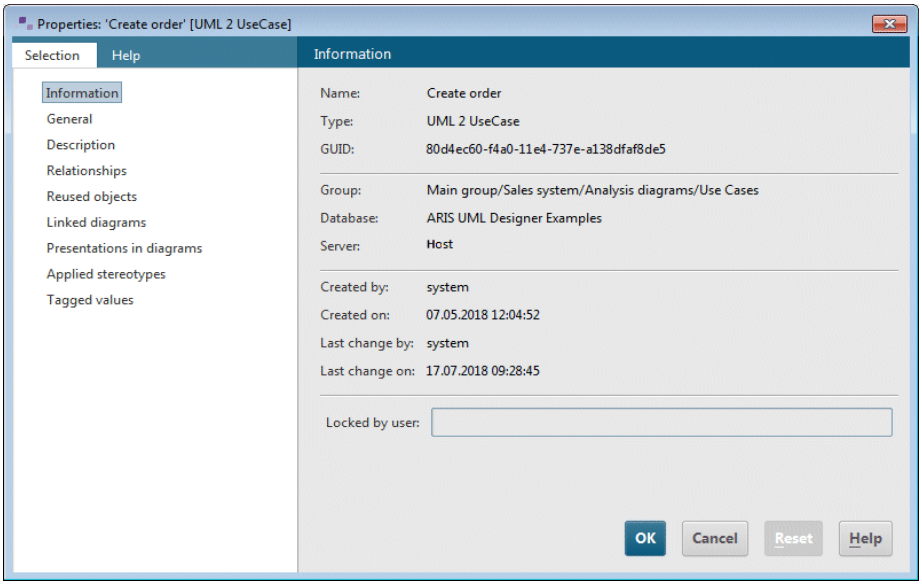


Figure 64: Properties dialog for a UML element

When you open the Properties dialog for an element or diagram, it essentially contains the same properties pages that are displayed on the **Explorer** tab. In addition, there is a properties page here for editing the description of the element or the diagram (see Figure 65).

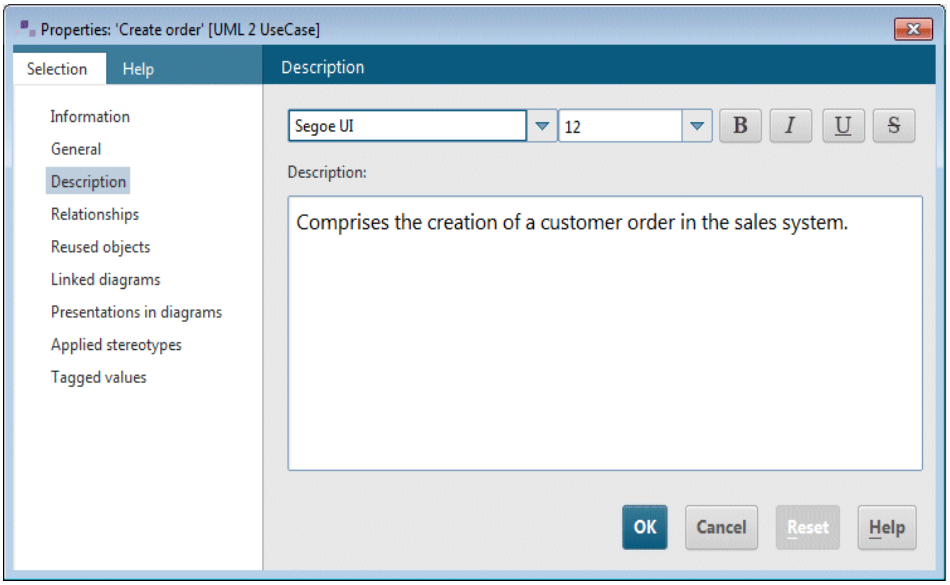


Figure 65: Properties page for displaying and editing the element description

3.2.4 Creating new elements in Explorer

New elements can be created in the Explorer tree by calling up the **New model element** option in the pop-up menu. This opens a submenu containing the element types that can be created within the selected element. Figure 66 shows this pop-up menu for a group. The four types **Group**, **Model**, **Package**, and **Profile** are available.

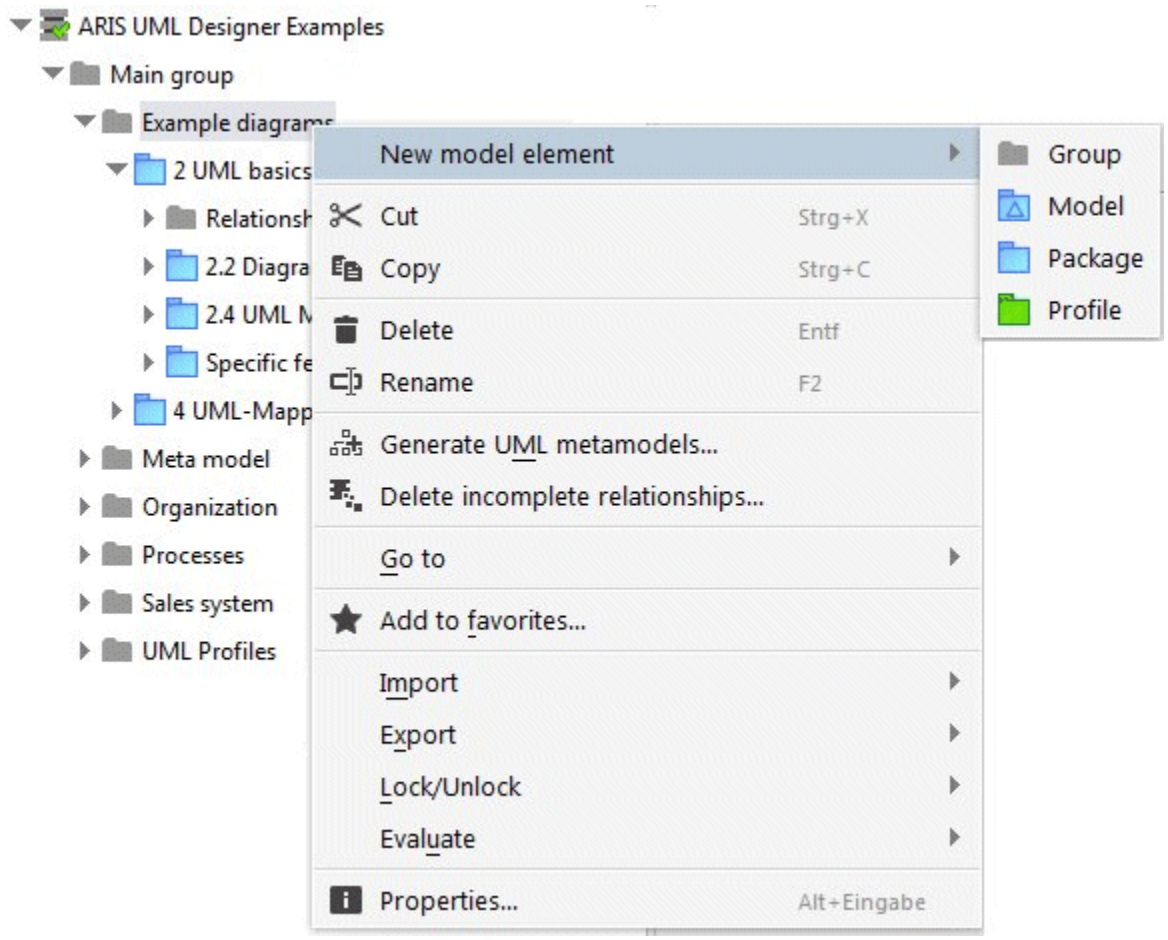


Figure 66: Pop-up menu for creating a new element in a group

If you call up this pop-up menu for a UML element, the number of possible element types that you can create within the selected element may exceed the capacity of the submenu. In this case, only the most important element types are provided directly in the submenu. As the maximum number of pop-up menu items depends on the screen size and resolution, you can specify the maximum number of items that this kind of pop-up menu can contain in the global options for ARIS UML Designer[21].

[21] ARIS > Options > UML > Explorer > Configure menu

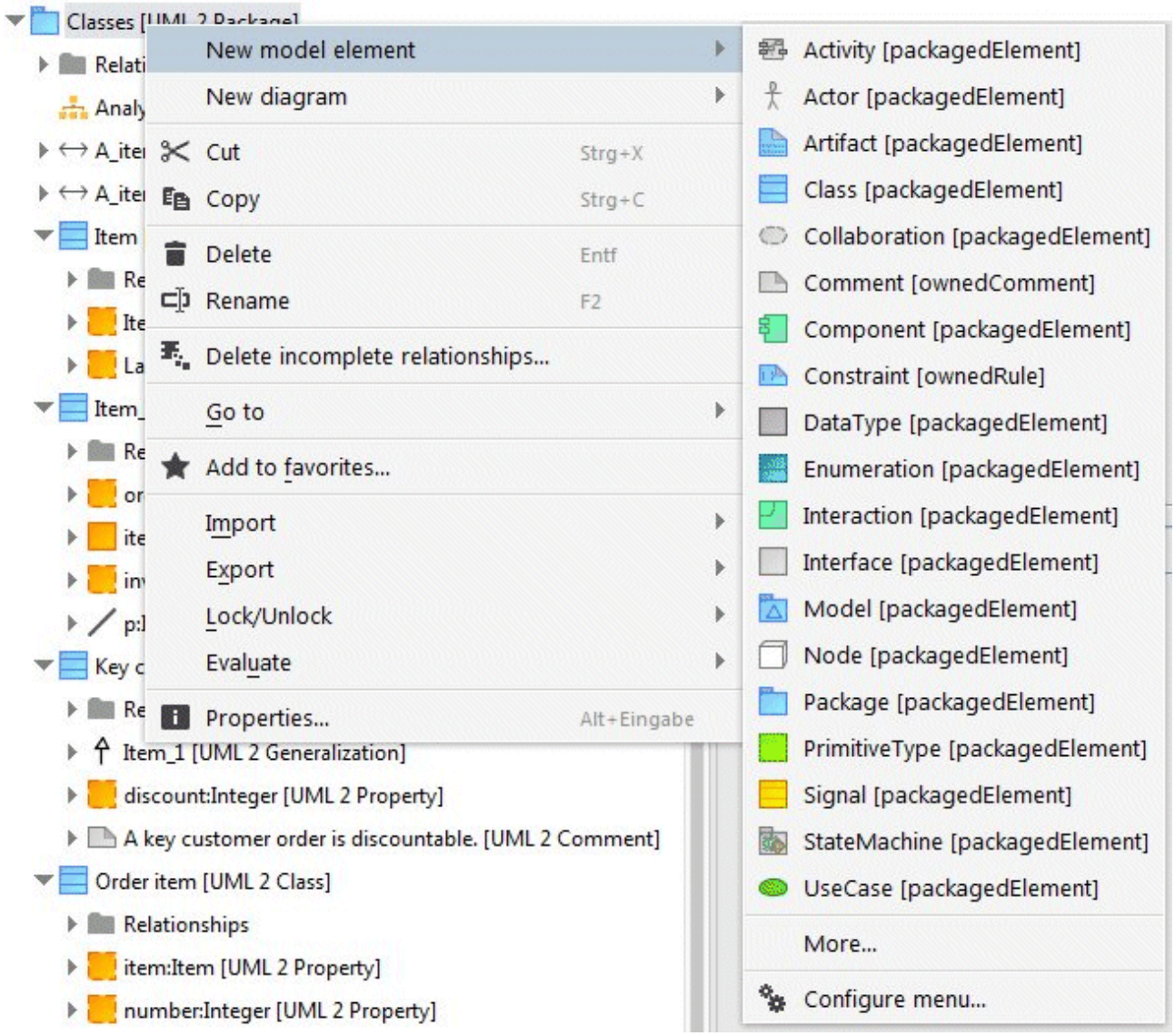


Figure 67: Pop-up menu for creating a new element in a UML package

Figure 67 shows the pop-up menu for creating a new UML element in a UML package. In addition to the element type, each menu item also contains the property under which the new element is created in the superior element. In this example, most elements would be created as packagedElement in the package (see also Figure 28: **Namespaces** diagram from the UML specification).

Clicking **More...** opens the **Create element** dialog, which lists all UML types that can be created within a package but are not included in the pop-up menu (see Figure 67).

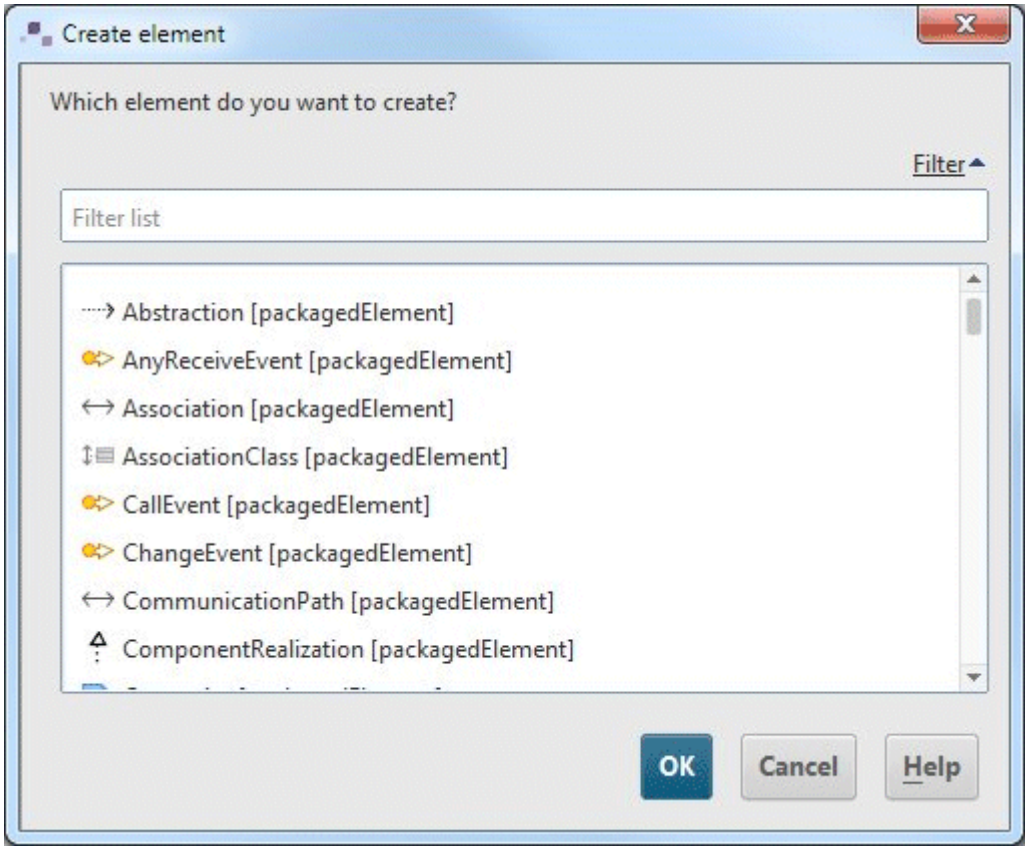


Figure 68: Dialog showing element types that are not included in the pop-up menu

The content of the **New model element** submenu can be individually adapted for each UML element type, allowing you to add the most important element types from a user perspective to the menu and to remove those that are not so important. Clicking the **Configure menu** item shown in Figure 67 opens the corresponding menu configuration dialog.

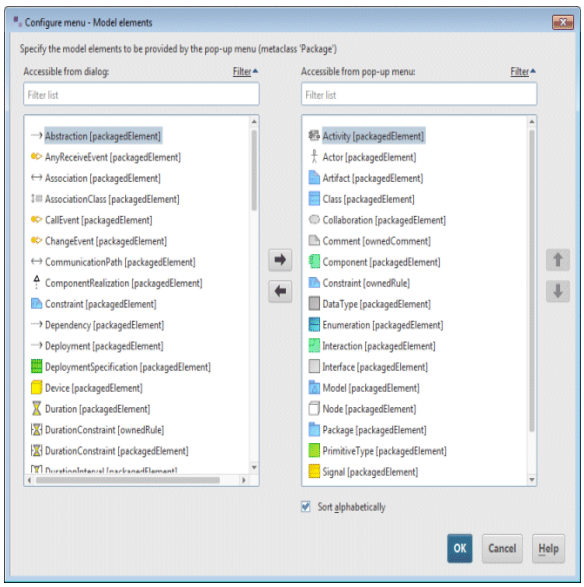


Figure 69: Dialog for configuring the **New model element** submenu

Figure 69 shows the dialog for configuring the **New model element** pop-up menu. The right-hand column contains all element types that are directly included in the pop-up menu, the left-hand column those that are available for selection in a dialog by selecting **More...**. If **Sort alphabetically** is enabled, the elements in the pop-up menu are sorted alphabetically. Otherwise, you can individually specify the order of the elements in the pop-up menu. Regardless of this setting, the element types in the dialog are always sorted alphabetically.

If you have a created a new element in Explorer, UML-specific text editing for the element is automatically activated.

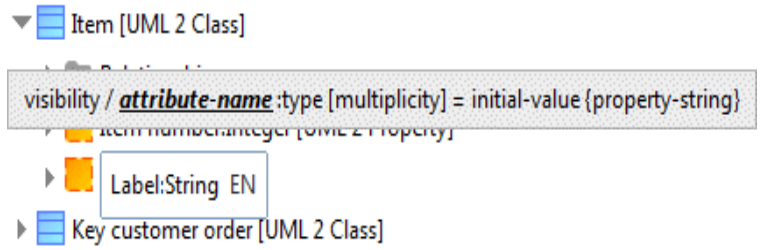


Figure 70: Text editing for a UML element in Explorer

The syntax help highlights the area to which the current cursor position in the text box relates. If the current text relates to a different UML element, all UML elements already loaded from the database whose name begins with the text entered and which are of the matching type are shown in a selection list:

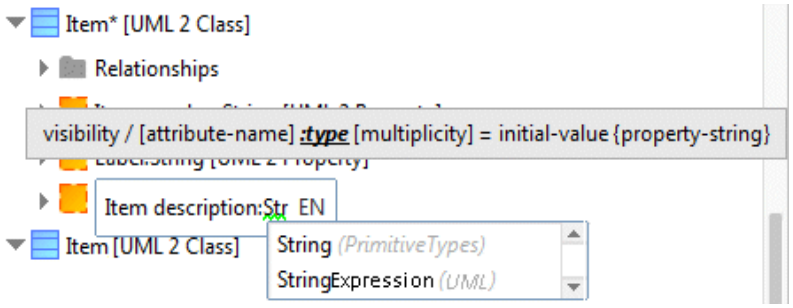


Figure 71: Selection list with matching elements in text editing

If a UML element is referenced that does not yet exist or has not yet been loaded from the database, its name is underlined with a green wavy line:

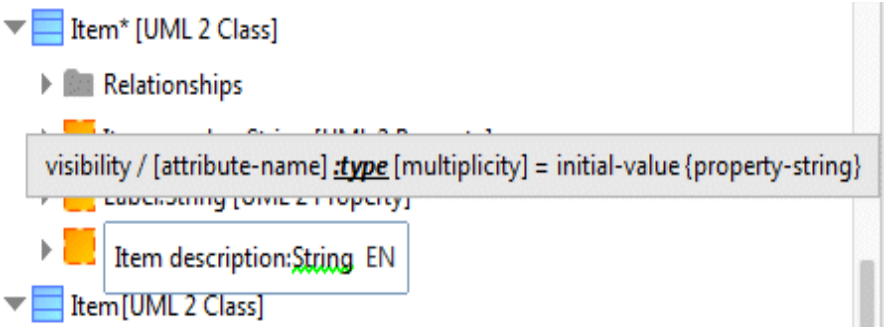


Figure 72: Unknown UML element in text editing

If you exit text editing in this situation, every unknown element is listed in the **Assign reference** dialog and you are offered the option of creating new elements or searching the database for matching elements.

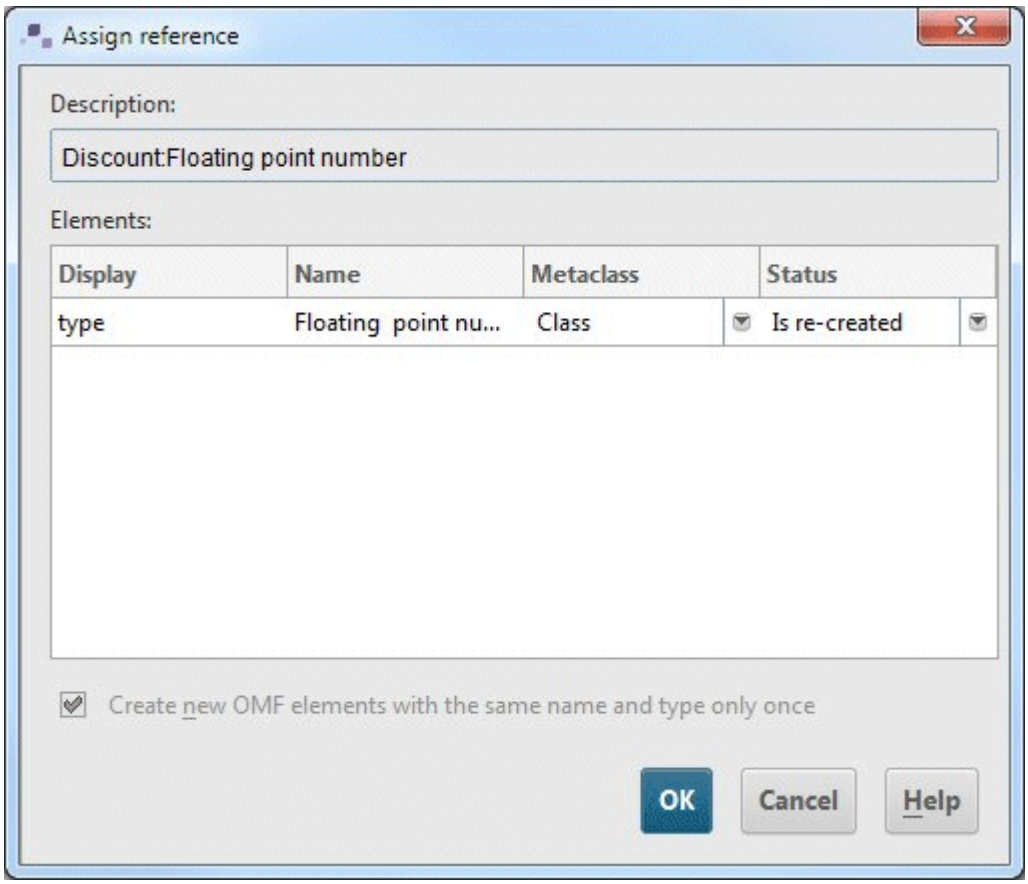


Figure 73: Dialog with unknown references in text editing

With no further entries in the dialog shown in Figure 73, clicking **OK** creates a new class with the name **String** as the type for the **Item description** attribute.

In the **Metaclass** column you can specify the exact type if a new element is to be created.

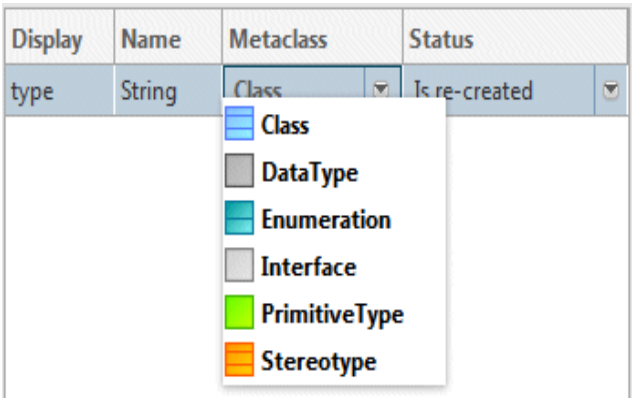


Figure 74: Selecting the type for the new element to be created

The **Status** column describes whether a new element is created or whether an existing element from the database is assigned. Clicking the button on the far right allows you to select from both options.

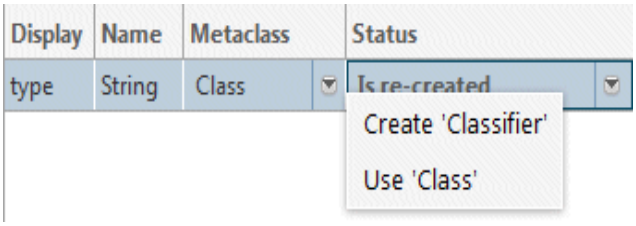


Figure 75: Selection options for creating or using a UML element

Selecting **Use** opens the **Select elements** dialog for searching for the element in the database.

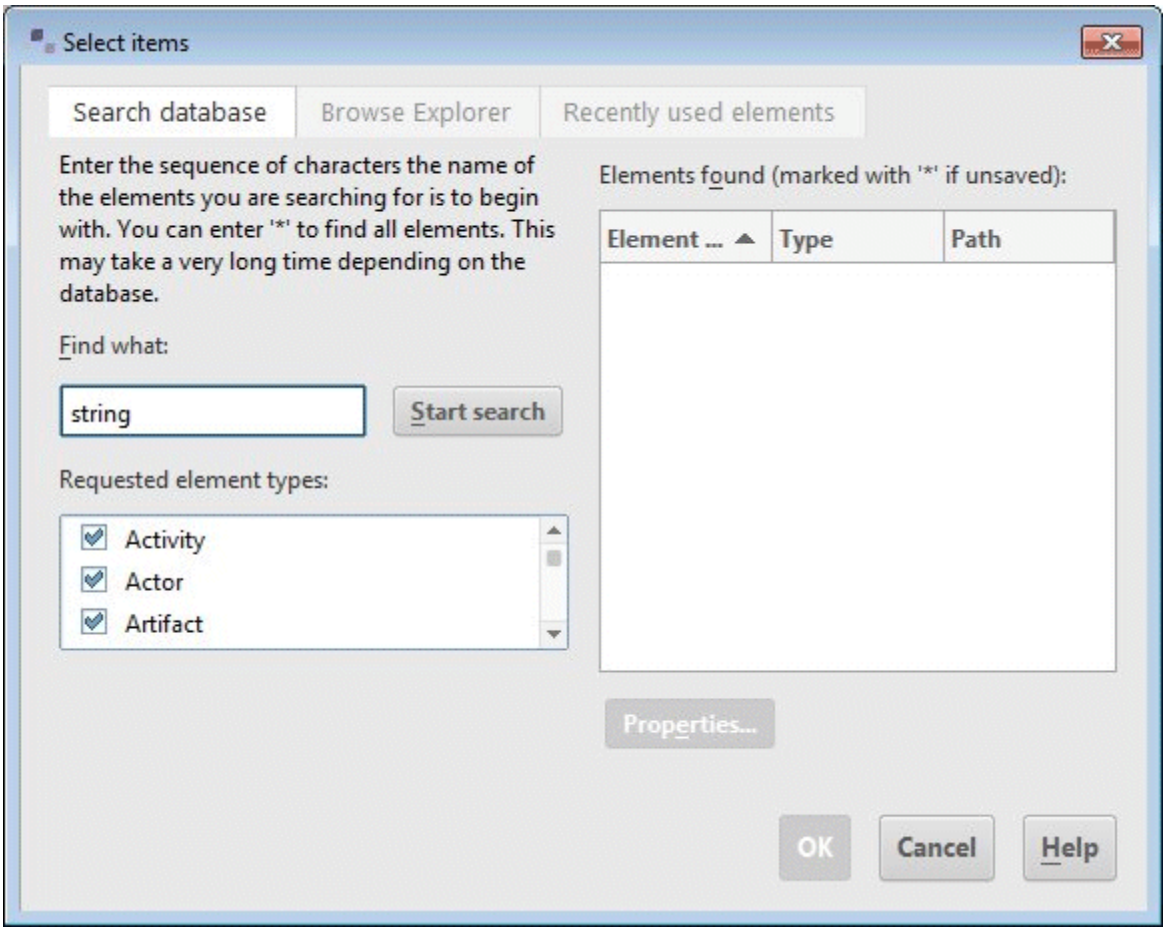


Figure 76: Search dialog

3.2.5 Creating new diagrams in Explorer

UML diagrams are always contained in a UML element. It is not possible to create a UML diagram directly within an ARIS group. UML elements of the **Package**, **Model**, **Profile**, or **Class**[22] type can contain any structure diagrams. Behavior diagrams, on the other hand, are always contained in the element represented, i.e., state diagrams in state machines, interaction diagrams[23] in interactions, and activity diagrams in activities.

New diagrams can be created in the Explorer tree by calling up the **New diagram** item in the pop-up menu.

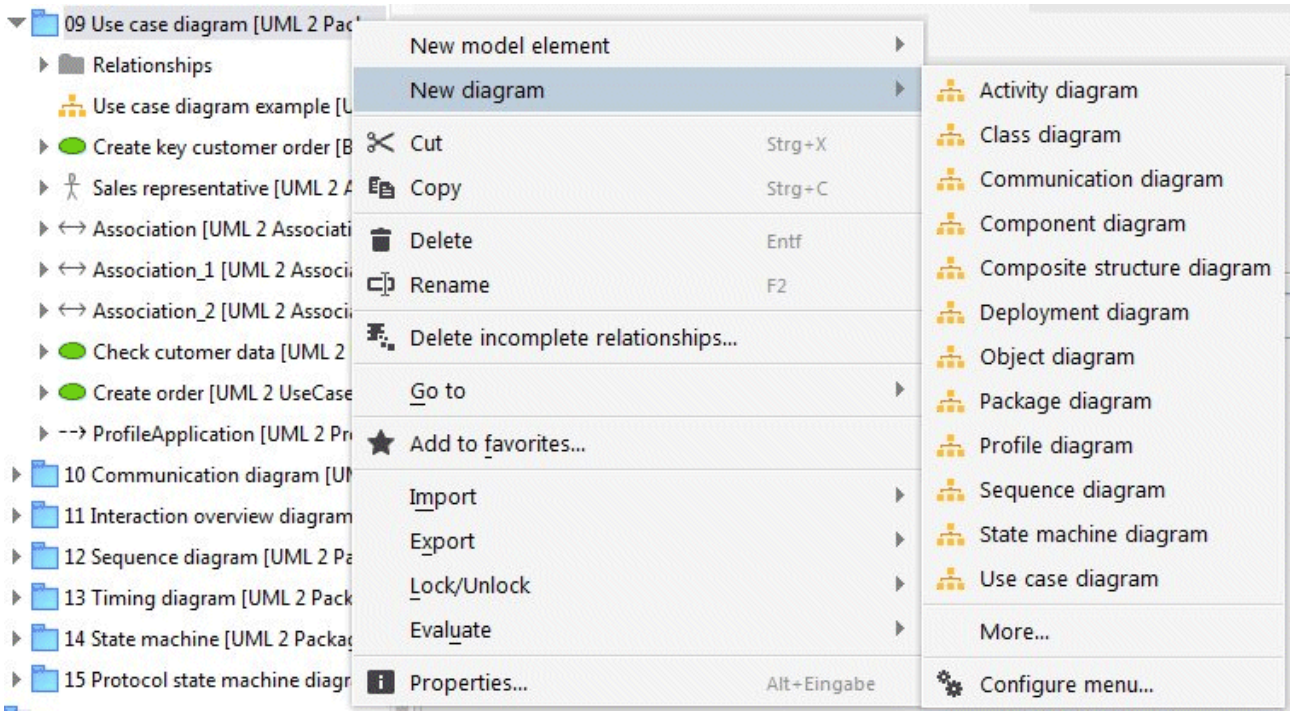


Figure 77: Pop-up menu for creating new UML diagrams

Figure 77 shows the pop-up menu for creating a new diagram in a UML package. The pop-up menu provides the same configuration options as the pop-up menu for creating new elements.

[22] This also applies to all elements whose metaclass is derived from Class, e.g., Component or AssociationClass.

[23] Communication, Sequence, and Timing diagrams are referred to as interaction diagrams.

3.3 Designer

When you open a UML diagram, it is displayed in the Designer component.

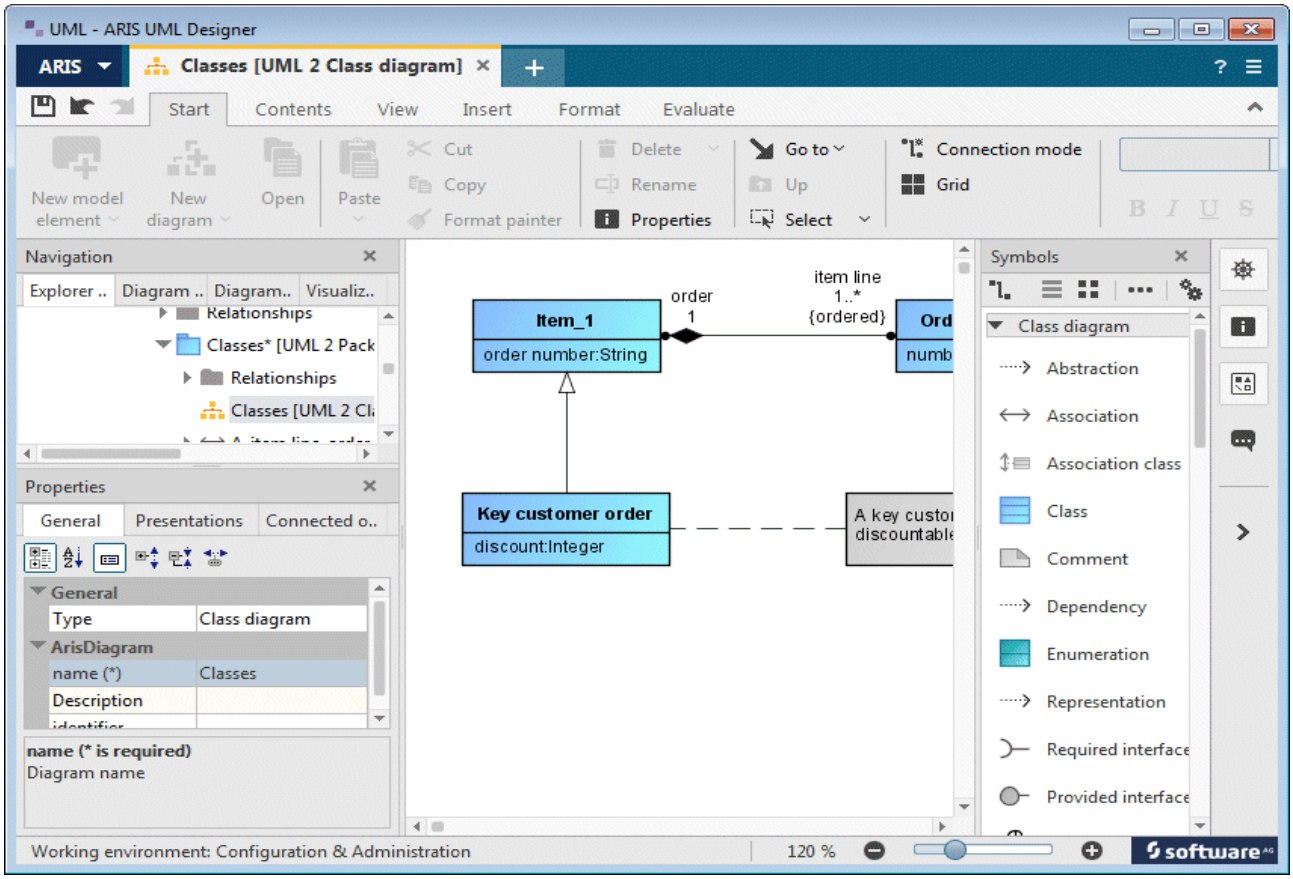


Figure 78: Designer component

The Designer component is divided into four areas – the modeling area, and the **Navigation**, **Properties**, and **Symbols** bars. The bars can be hidden to create more space for the diagram representation. In addition, you can rearrange the bars by dragging them with the mouse button held down.

The following buttons are available on the far right of the window:

 Shows and hides the **Navigation** bar

 Shows and hides the **Symbols** bar

 Shows and hides the **Properties** bar

 Shows and hides the **Implicit changes** bar  Hides all bars

 Shows all bars

For the screenshots shown below, the individual bars have been arranged in such a way that they provide a useful representation adapted to the page width available in this document.

3.3.1 Navigation bar

In addition to the two trees familiar from the **Explorer** tab, the **Navigation** bar contains the diagram overview of the list of elements that appear in the diagram.

3.3.1.1 Diagram overview

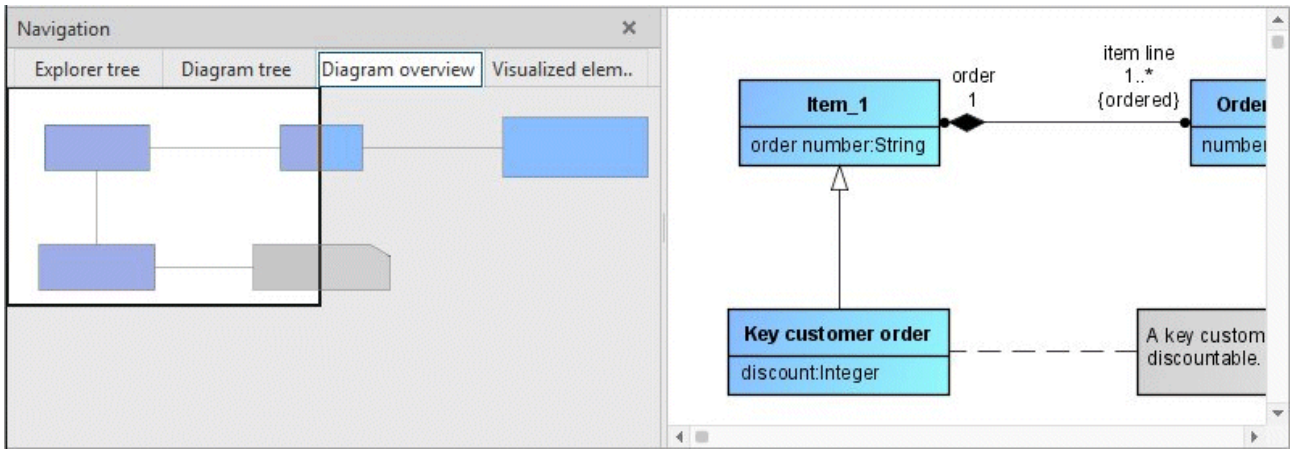


Figure 79: Diagram overview

The Diagram overview provides a schematic view of the entire diagram and indicates the section that is visible in the modeling area with a white rectangle. By moving the rectangle, you can move sections of the diagram that are not visible in the modeling area into the visible area.

3.3.1.2 Visualized elements

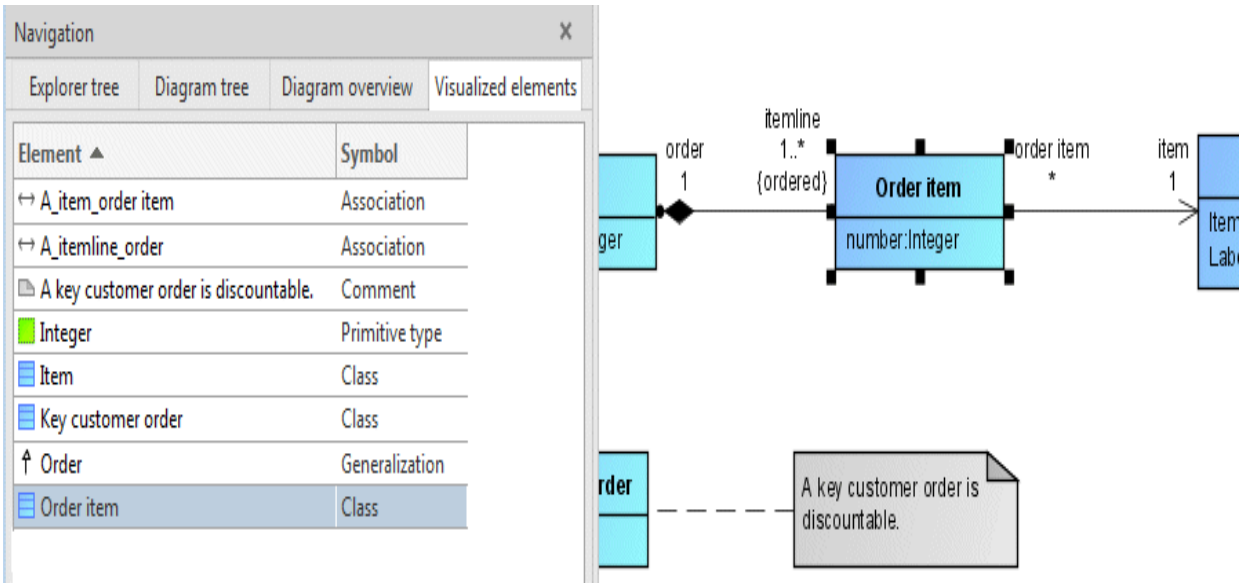


Figure 80: Visualized elements

The **Visualized elements** page shows a list of all the UML elements that appear in the diagram. Selecting an element in the list also selects the element in the diagram and moves the section shown in the modeling area so that this element is visible.

3.3.2 Properties bar

The **Properties** bar essentially shows the same properties pages as the **Explorer** tab. However, as the available space is smaller you can specify which properties pages are to be displayed in the Designer component in the global options.

In each case, the properties of the element currently selected in the **Navigation** bar or in the diagram are displayed.

3.3.2.1 Format

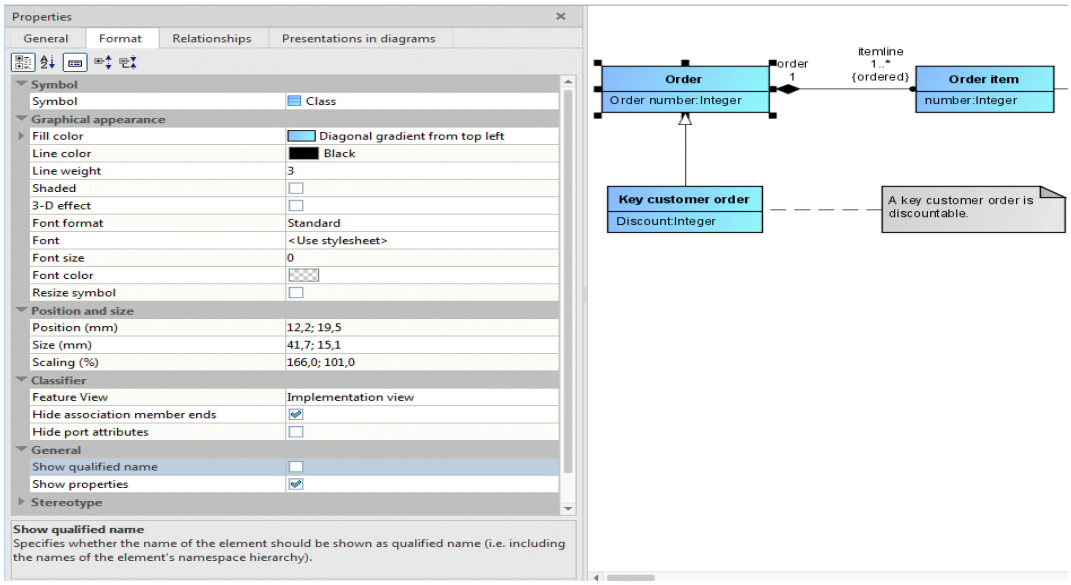


Figure 81: **Format** properties page for the element selected in the diagram

The **Format** properties page is displayed for elements in diagrams. It shows all representation options for the selected element. If multiple elements are selected, it shows the combined representation options for the selected elements. This enables the graphical representation of multiple elements to be edited simultaneously.

Just as for the general properties, a brief description of the selected representation option is displayed at the bottom and you can choose between thematic grouping and an alphabetical display of the representation options.

These representation options include both general graphical properties such as colors, line weight, or font format, and UML-specific options that specify which details the relevant elements are to display in the diagram.

3.3.3 Symbols bar

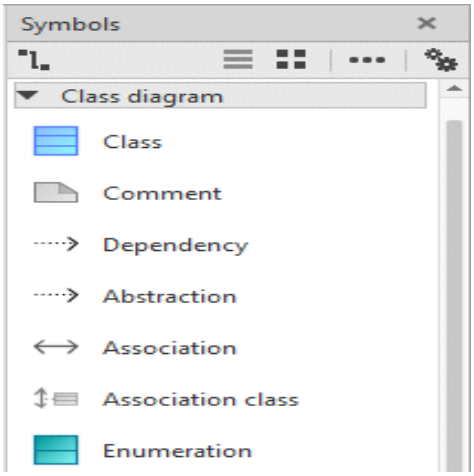


Figure 82: **Symbols** bar for a Class diagram

The **Symbols** bar is used to create elements and relationships in the diagram. In contrast to ARIS Architect and ARIS Designer, it also contains edge symbols.

If you hover the mouse pointer briefly over a symbol, a description of the symbol is displayed.

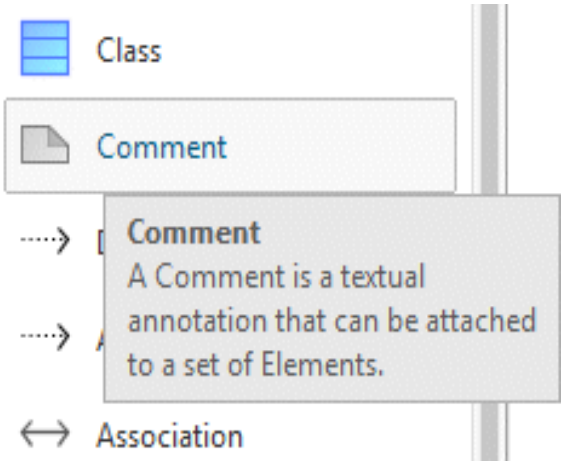


Figure 83: Symbol description

You can use the **Show symbols with names** and **Show symbols without names** buttons to show and hide the symbol names in the **Symbols** bar.

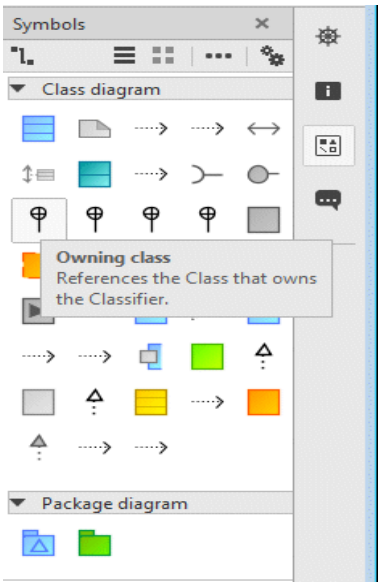


Figure 84: **Symbols** bar with symbol names hidden

Selecting the **Remove symbol** item in the pop-up menu enables a symbol to be removed from the **Symbols** bar.

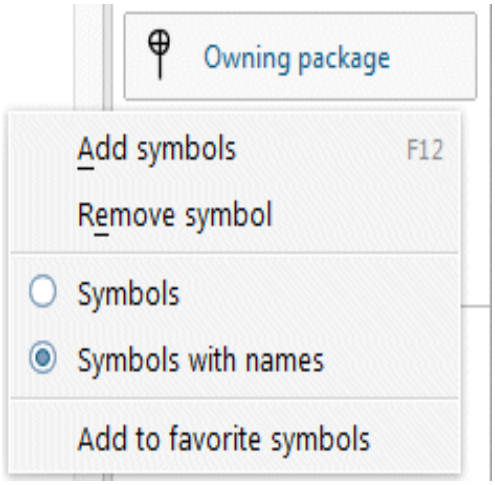


Figure 85: Pop-up menu in the **Symbols** bar

You can click the **Add symbols** button or the **Add symbols** item in the pop-up menu to open the dialog for configuration of the **Symbols** bar.

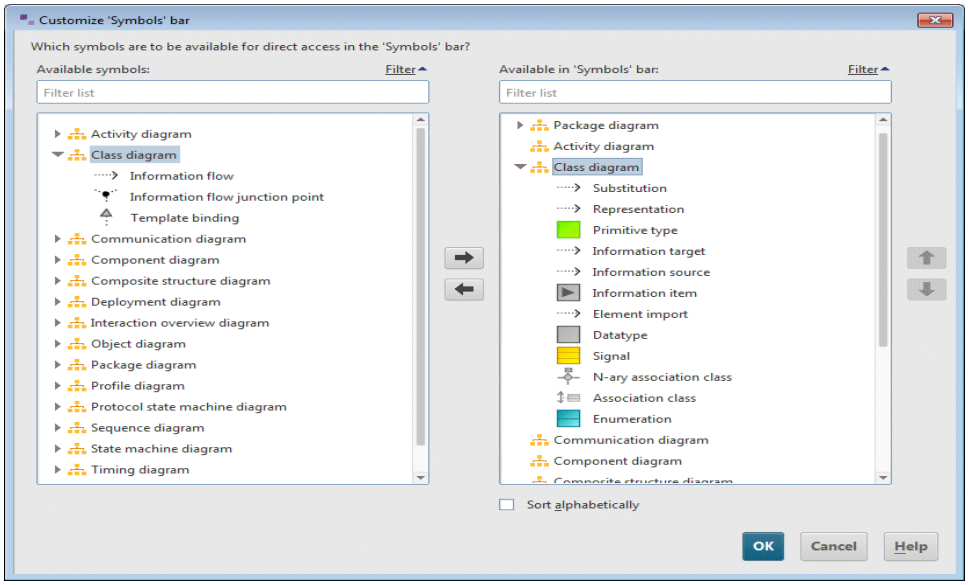


Figure 86: Dialog for configuration of the **Symbols** bar

The dialog contains a list of the available symbols and a list of the symbols contained in the **Symbols** bar, in each case grouped by diagram type. In the case of structure diagrams such as the Class diagram, you can add any UML diagram types to the **Symbols** bar.

A text input box above the list enables the list to be filtered by the symbols whose name contains the text entered. The corresponding diagram nodes are automatically expanded in the list, allowing fast access to the symbol you are looking for (see Figure 87).

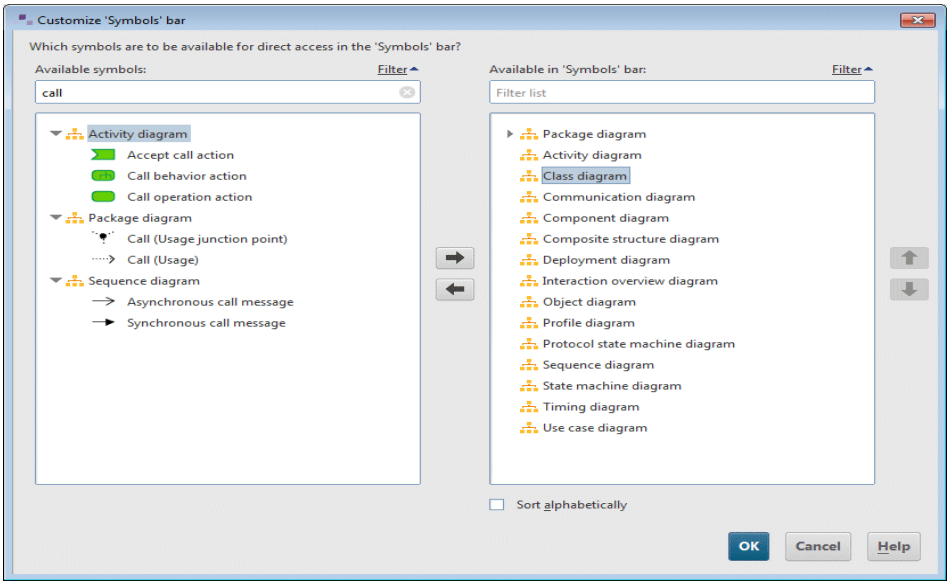


Figure 87: All available symbols whose name contains the text **Call**

In addition, the **Symbols** bar in ARIS UML Designer provides the option of grouping frequently used symbols in the upper section of the **Symbols** bar for fast access, regardless of their diagram type.

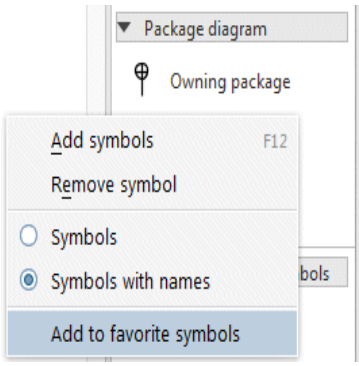


Figure 88: **Add to favorite symbols** pop-up menu item

To do this, select the **Add to favorite symbols** pop-up menu item for the corresponding symbol in the **Symbols** bar.

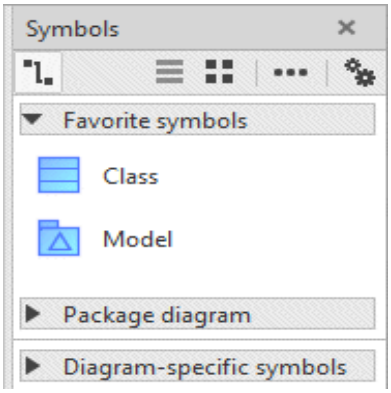


Figure 89: Favorite symbols

The symbol is then also displayed in the top section of the **Symbols** bar under **Favorite symbols**.

Clicking **Create additional symbol presentation** enables you to create an element in a diagram whose symbol is not to be permanently contained in the **Symbols** bar for all diagrams of the same type. The Create presentation dialog opens (see Figure 90).

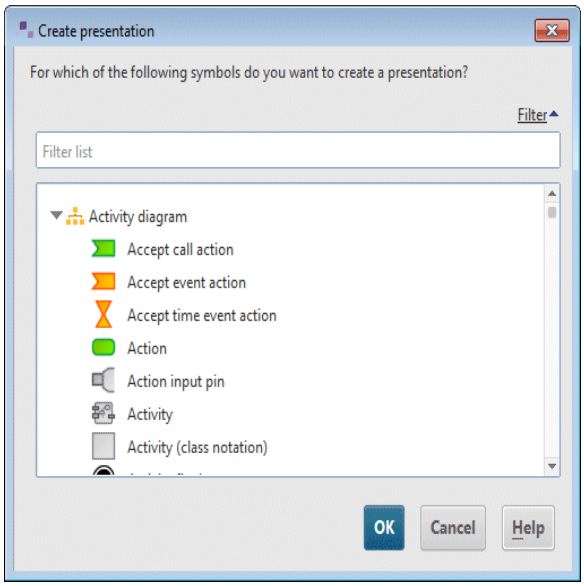


Figure 90: Dialog for selecting the symbol

The dialog lists all available symbols, once again grouped by diagram type, and provides the same filtering by text input as the dialog for configuration of the **Symbols** bar.

If a diagram includes presentations of symbols that are not contained in the **Symbols** bar for the diagram type, they are displayed in the **Symbols** bar for the relevant diagram in the **Diagram-specific symbols** section of the **Symbols** bar.

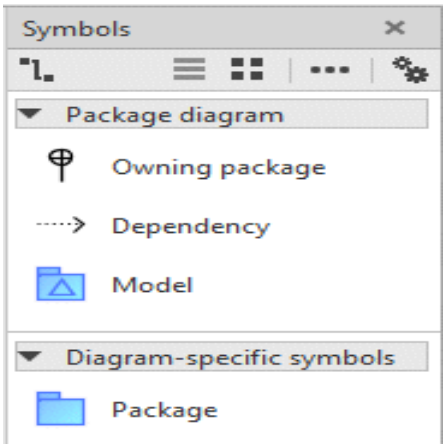


Figure 91: Diagram-specific symbols

3.3.4 Implicit changes bar

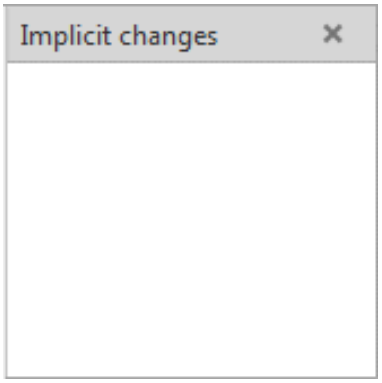


Figure 92: Implicit changes bar

The **Implicit changes** bar logs changes to UML elements and diagrams that occur implicitly due to changes elsewhere.

A typical example of this kind of implicit change is shown below.

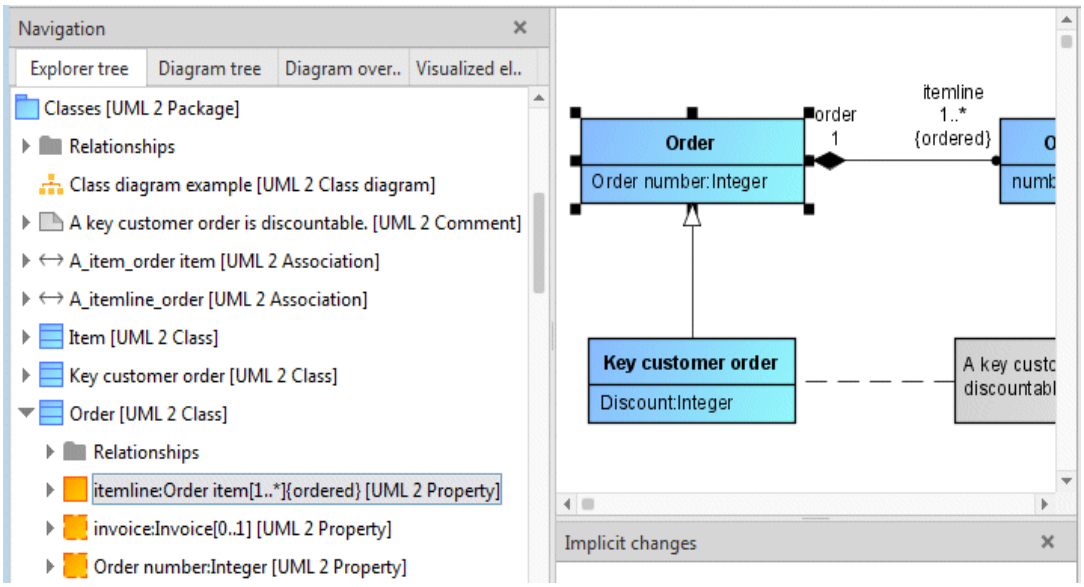


Figure 93: Binary associations whose ends are attributes of the classes involved

Figure 93 shows a Class diagram with a binary association between the **Order** and **Order item** classes. The two ends of the association are navigable as attributes of the classes involved. For this reason, the **itemline** association end is a child of the **Order** class in the Explorer tree. It is displayed as text **itemline:order item[1..*]{ordered}** with its type, the **Order item** class.

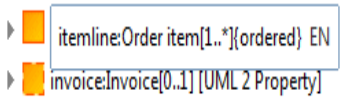


Figure 94: Renaming the association end and deleting the type specification

If we now rename this association end and delete the type specification, this means that the **itemline** association end is no longer linked to the **Order item** class. This also removes the association edge from the diagram, which represents an implicit change that is not always directly identifiable for the user.

This implicit change is logged in the **Implicit changes** bar.

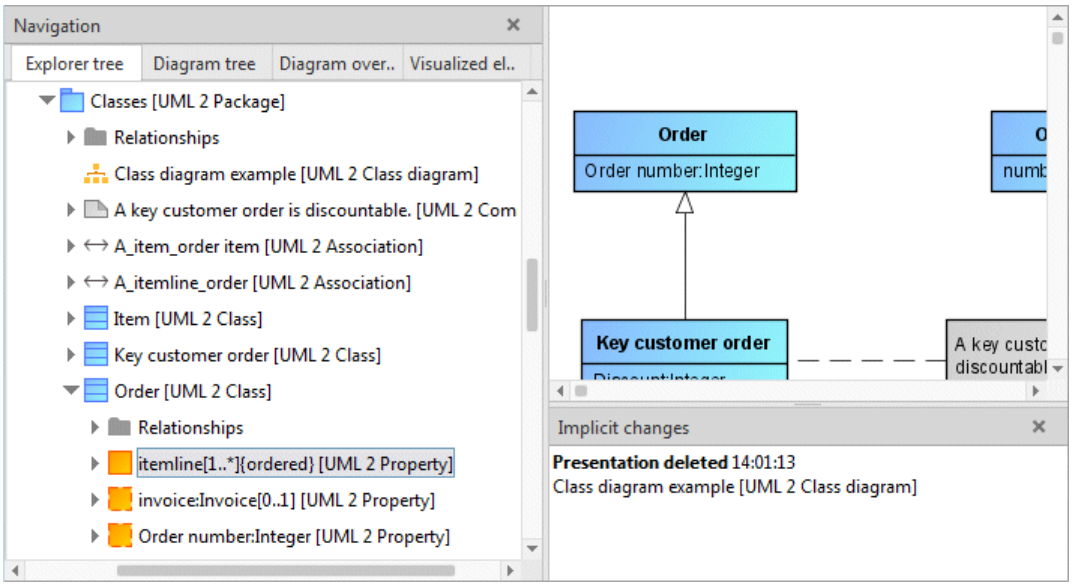


Figure 95: Logged implicit change

Logging of implicit changes can be enabled and disabled on the global options page **UML -> General**.**Modeling**

Essentially, graphic modeling in ARIS UML Designer is based on the same principles as in ARIS Architect and ARIS Designer. For this reason, this section focuses on the modeling-specific special features of ARIS UML Designer, with only a brief discussion of the principles of graphic modeling in ARIS.

3.3.4.1 Creating new node presentations

New node presentation elements can be created in the diagram by first selecting the corresponding symbol in the **Symbols** bar by clicking it.

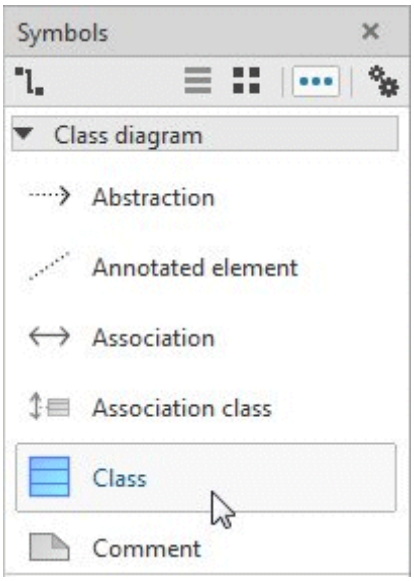


Figure 96: Selecting the **Class** node symbol

If you then move the mouse pointer to the modeling area, a preview of the new presentation to be created is displayed at the mouse pointer.

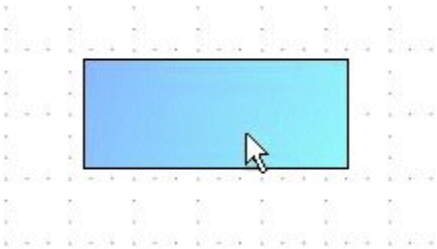


Figure 97: Mouse pointer with preview of a class presentation

Clicking in the modeling area creates the presentation at that point and displays a text input box for entering the name.

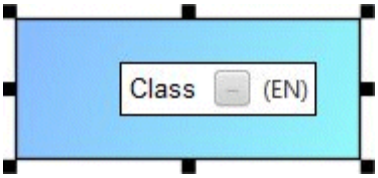


Figure 98: New class placed with text input box for the name

This text input box initially contains the corresponding symbol names, possibly supplemented by an underscore and a number (**_1**,**_2**, etc.) if elements of the same type with this name already exist.

The content of the text input box is applied when you complete your entry by pressing the Enter key or by clicking at any point in ARIS UML Designer outside the text input box. Depending on your setting in the global options, text entry can also be completed by simultaneously pressing Ctrl and Enter[24]. If, on the other hand, you exit the entry by pressing **Esc**, the original name is retained.

If you have entered the name of an existing element of the same type in the text input box, the **Select element** dialog opens and asks whether you want to create a new presentation for the existing element in the diagram, or whether you want to create a new element with this name (see Figure 99).

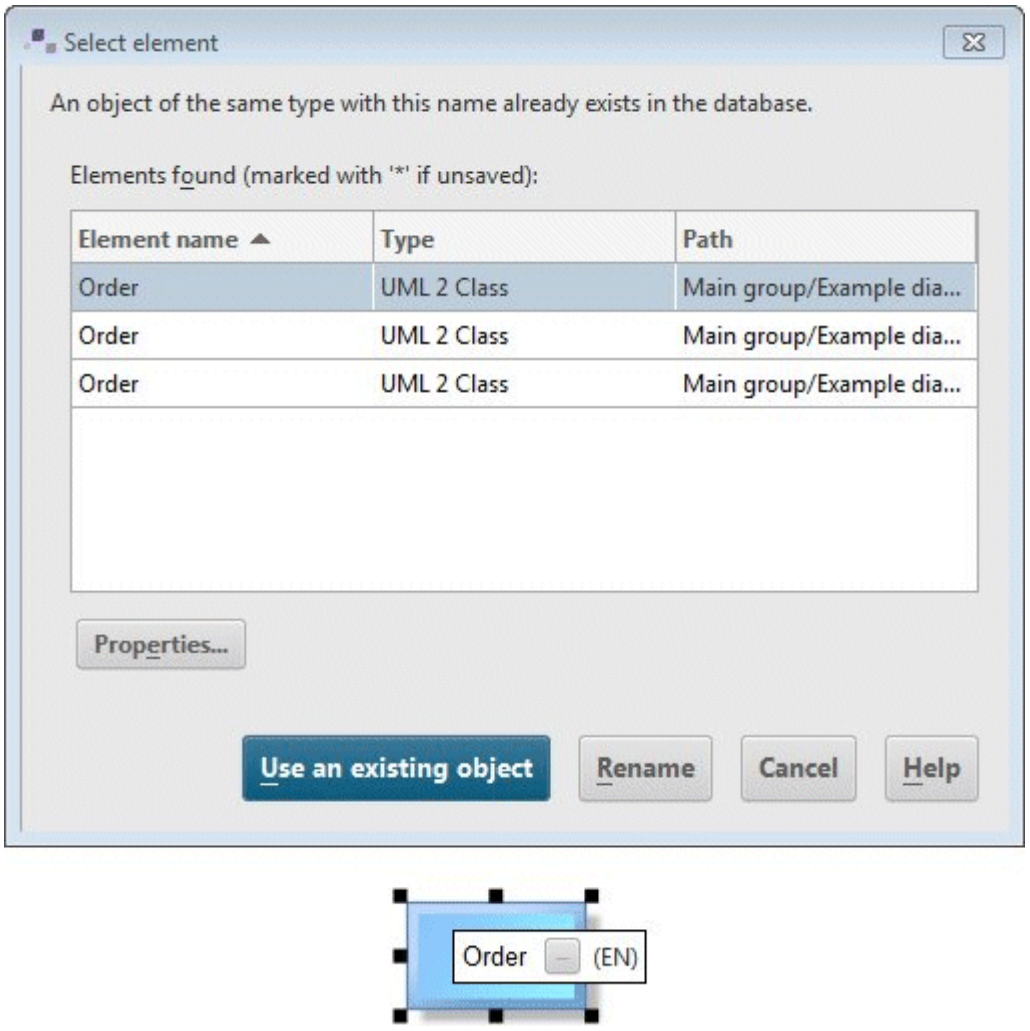


Figure 99: Create dialog after entering the name of existing elements

[24] See ARIS > Options > UML > Designer - General > Use Enter for line break

You can also create a new presentation for an existing element by dragging the element from the Explorer tree to the modeling area or by copying it from the Explorer tree to the clipboard and pasting it in the diagram. If several possible symbols exist that can represent the element in the diagram, you are asked which of these symbols you want to use (see Figure 100). By contrast, if you copy a presentation of the element from a diagram to the clipboard and then paste it into a diagram, there is no prompt as in this case all the presentation properties of the original, such as symbol, color, and size are also copied.

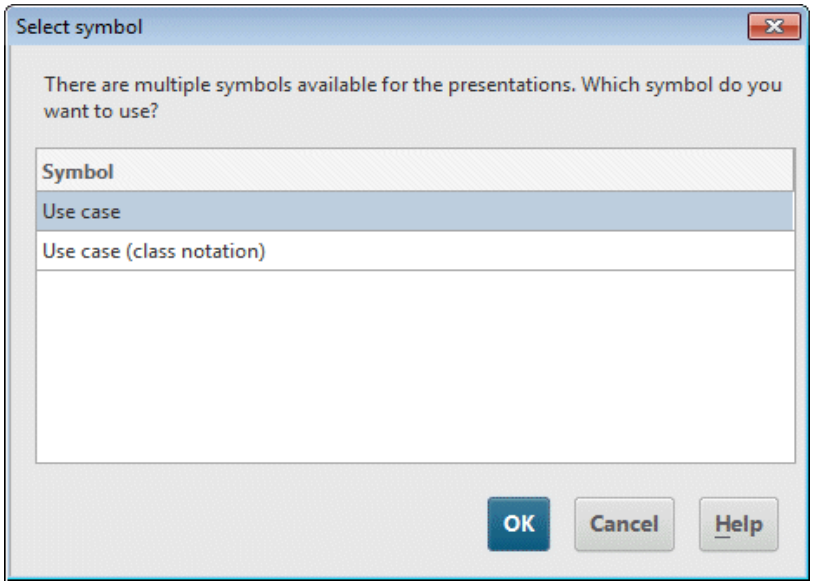


Figure 100: Dialog asking which symbol is to be used

3.3.4.2 Creating a new edge presentation

To create a new edge presentation, first select the corresponding edge symbol in the **Symbols** bar.

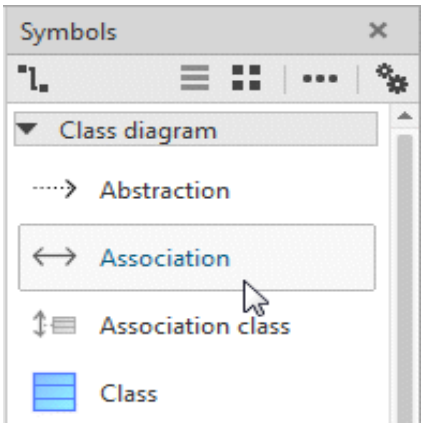


Figure 101: Selecting the **Association** edge symbol

Moving the mouse pointer to the relevant source element in the diagram graphically displays the anchor point closest to the mouse pointer.

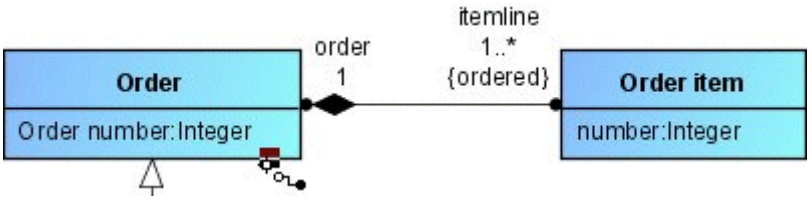


Figure 102: Displaying the edge anchor point on the source element

Clicking at this point specifies the starting point of the edge. Starting from this point, a preview of the edge up to the current mouse pointer position is then displayed. If you move the mouse pointer to the relevant target element, the nearest possible anchor point is once again displayed graphically.

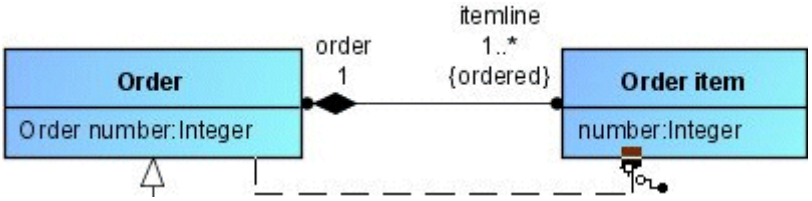


Figure 103: Displaying the edge anchor point on the target element and edge preview

Clicking the target element creates the edge.

If a relationship of the same type already exists between the source and target element and more than one relationship of this type is allowed between the two elements, a dialog appears to ask whether you want to create a new edge presentation for the existing relationship or to create a new relationship (see Figure 104).

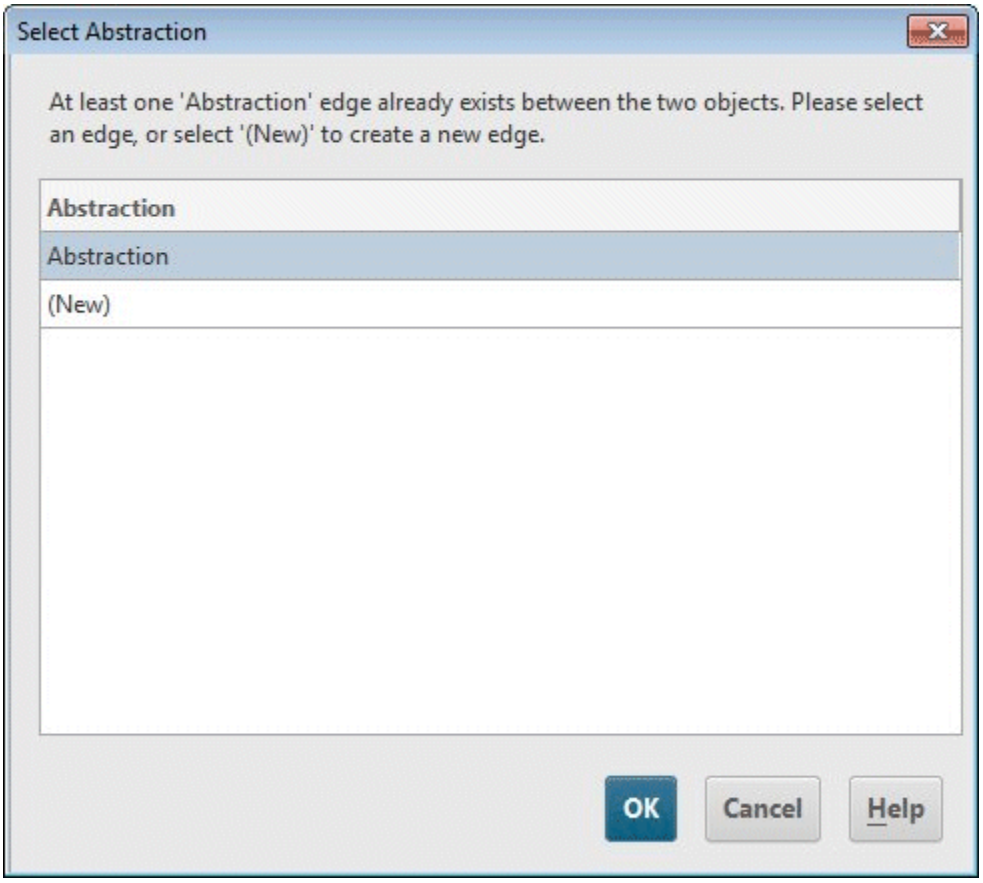


Figure 104: Dialog for creating an association between two classes between which an association already exists

As an alternative to selecting a particular edge symbol, you can also use the general edge symbol from the **Symbols** bar to create an edge in the diagram.

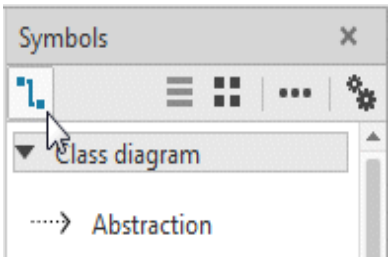


Figure 105: General edge symbol

In this case, when creating the edge after clicking the target element a selection list is displayed, in which you are prompted to select the specific edge type.

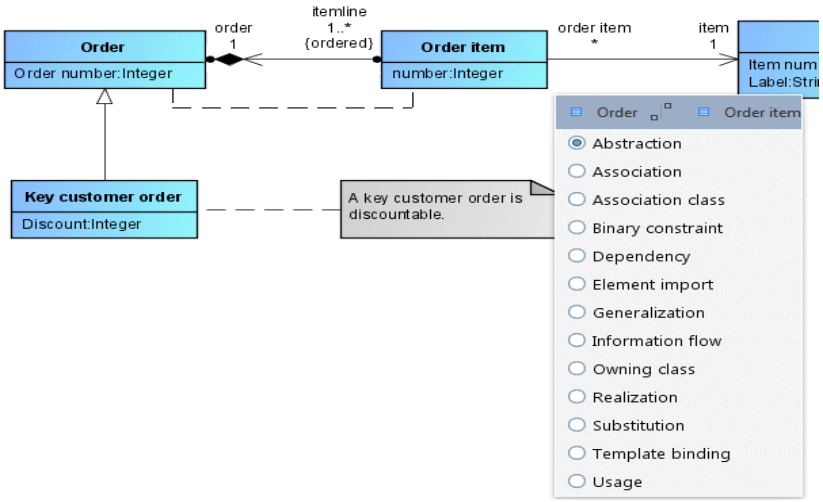
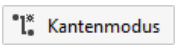


Figure 106: Selection list with edge types

Clicking  in the **Start** tab bar enables and disables edge mode. If edge mode is enabled, possible edge anchor points are automatically displayed in the diagram as soon as the mouse pointer is close to them. It is not then necessary to select the general or a specific edge symbol in the **Symbols** bar first.

3.3.4.3 Deleting presentations and elements

In ARIS UML Designer, the pop-up menu for a presentation contains two different items for deleting:

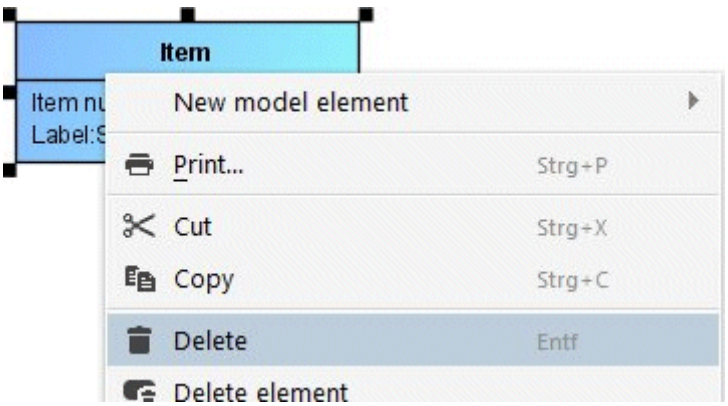


Figure 107: Delete functionalities in the pop-up menu

Delete only deletes the presentation in the diagram. The element or relationship it represents remain in the model, i.e., available in the database.

Delete element deletes not only the presentation but also the element itself or the relationship represented in the model.

3.3.4.4 Mini toolbar

Clicking a node presentation displays a small toolbar next to it. This toolbar can contain node and edge symbols and you can adapt it individually for each symbol.

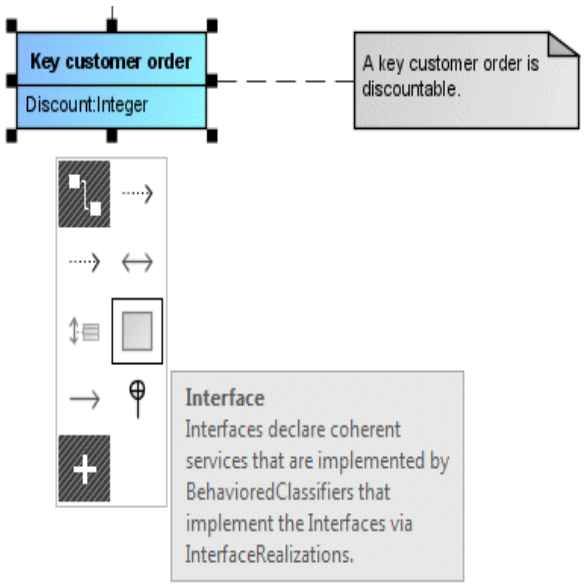


Figure 108: Mini toolbar

If you select a node symbol from the mini toolbar, you can create a corresponding element in the diagram, which is automatically linked to the selected node presentation by an edge. If several edge types are allowed between the two elements, a selection list is shown. In this case, first select the edge type and then click in the modeling are at the position where you want to create the node presentation.

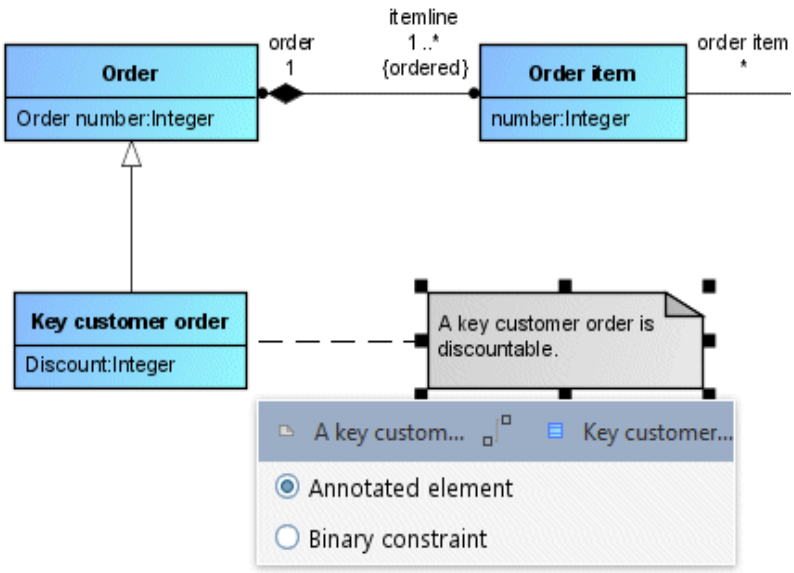


Figure 109: Node and edge preview with edge type selection after clicking the **Comment** symbol in the mini toolbar

When you select an edge symbol from the mini toolbar, the node presentation for which the mini toolbar is displayed is used as the source element for the edge and then you only need to click the target element in the diagram to create the edge.

You can remove a symbol from the mini toolbar by selecting **Remove symbol** in the pop-up menu. Selecting **Add symbol** or clicking **Add symbols** opens the dialog for adding a symbol.

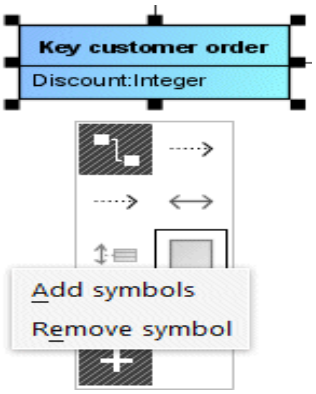


Figure 110: Pop-up menu in the mini toolbar

3.3.4.5 Modeling and hierarchy in Explorer

A series of edge types in ARIS UML Designer graphically represent the fact that an element is contained in another element.

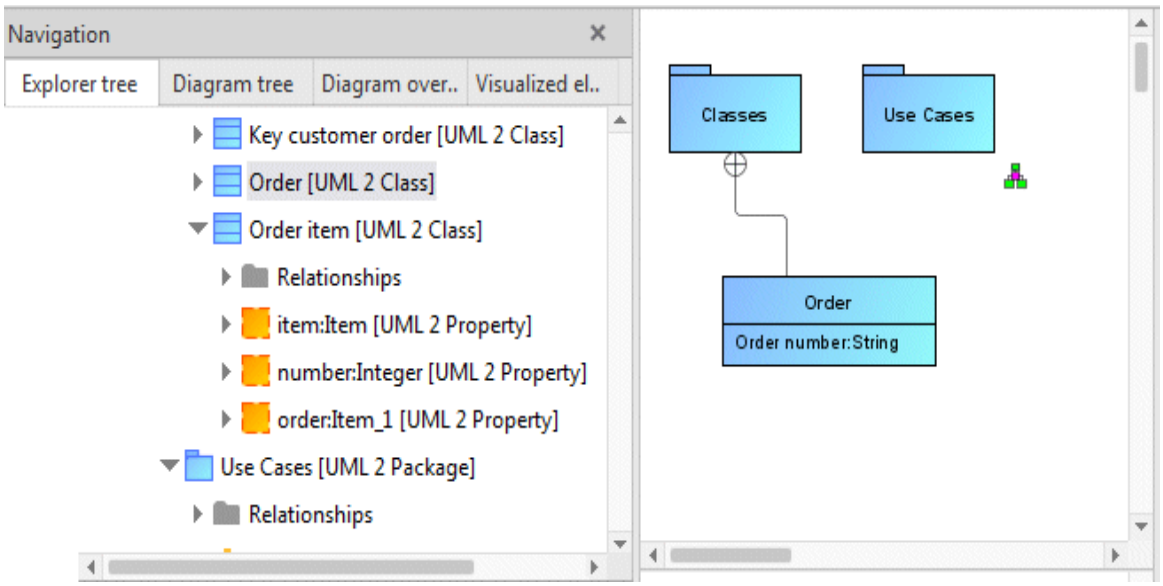


Figure 111: **Owning package** relationship between a class and the package in which it is contained

The diagram in Figure 111 shows this kind of relationship of the **Owning package** type between the **Order** class and the **Classes** package. This shows that the class is contained in the package, which you can also see from the hierarchy in the Explorer tree[25].

[25] See also Figure 28 in the Namespaces section.

What happens if you create an additional edge of this type from the **Order** class to the **Use cases** package is shown in Figure 112:

- The original relationship between the **Order** class and the **Classes** package is deleted.
- The **Order** class is moved to the **Use cases** package in the Explorer tree.

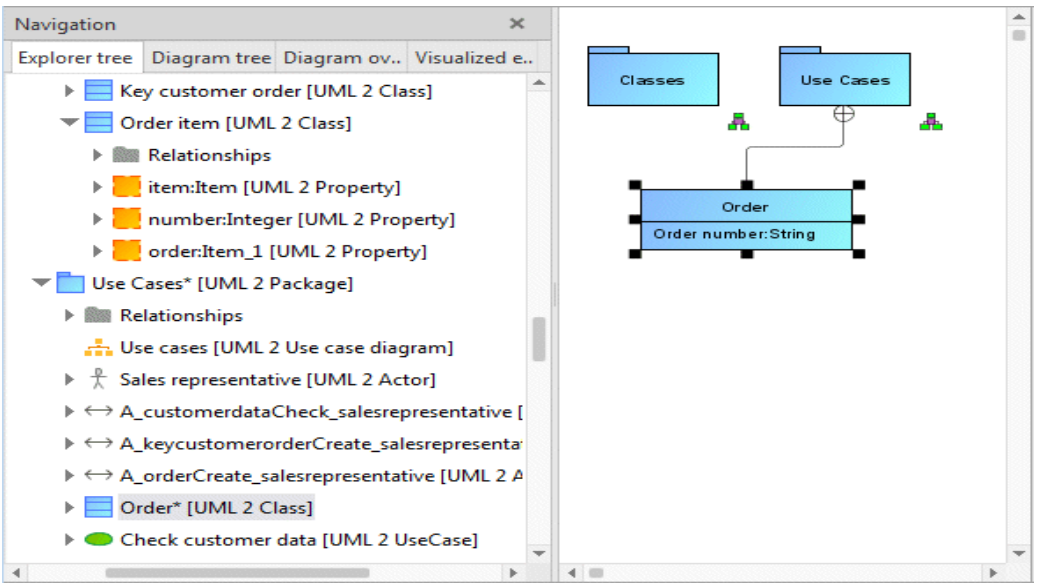


Figure 112: After creating a second Owning package relationship

3.3.4.6 Graphic nestings

In many cases, in ARIS UML Designer diagrams the ownership of one element by another can also be indicated by a presentation of the element being graphically nested in the presentation of the owning element. In some cases, graphic nesting can also represent a relationship that is not ownership. However, a common feature of all graphic nestings in ARIS UML Designer is that they always represent a relationship between the elements at definition level.

When creating or moving an element in the diagram, presentations in which the element can be nested are indicated by a border when the mouse pointer is located within the potential nesting container.

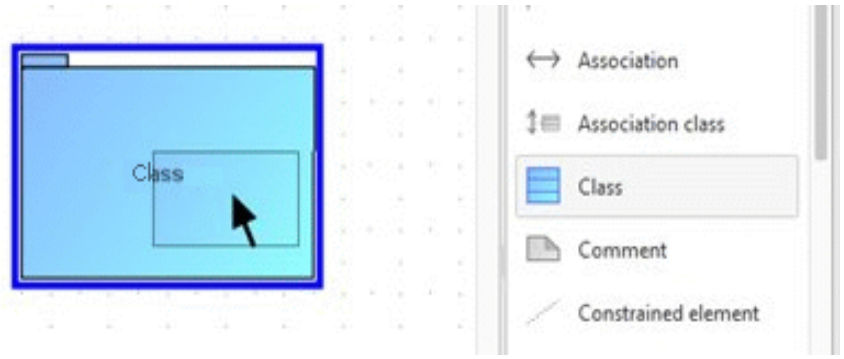


Figure 113: Graphic indication of a package as a potential nesting container when creating a new class

Once the class has been created as a nested presentation in the package, it is also contained in the package in the Explorer tree.

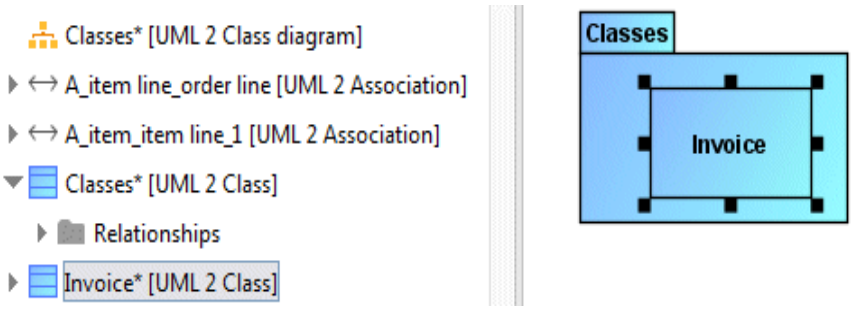


Figure 114: Class nested in a package

The appearance of a class in a diagram as an element nested in a package represents an alternative notation to the link using an Owning package edge shown in Figure 111.

A series of options exist that influence the behavior of ARIS UML Designer when modeling with nestings.

The first options[26] relate to how the relationships underlying the graphic nesting are handled.

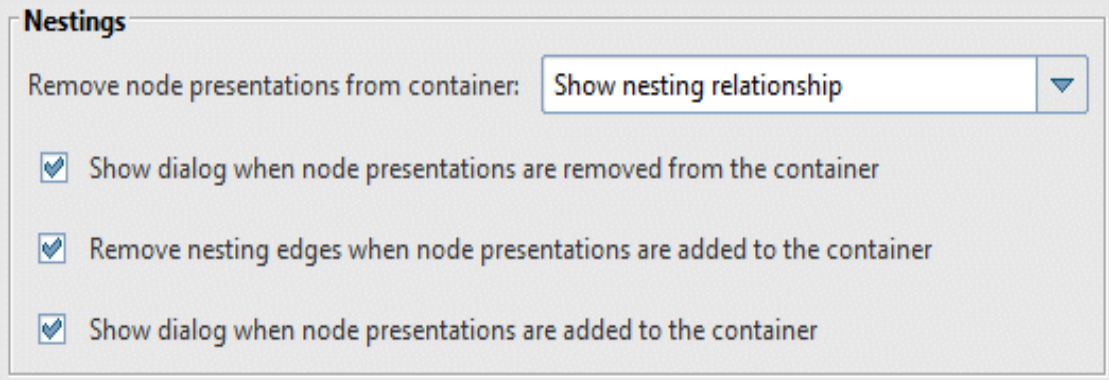


Figure 115: Options for nestings

If the two **Show dialog when...** options are enabled, a corresponding query is displayed when modeling. In this case, if you move the class in Figure 114 out of the package to the diagram background, the **Unnest node** dialog opens:

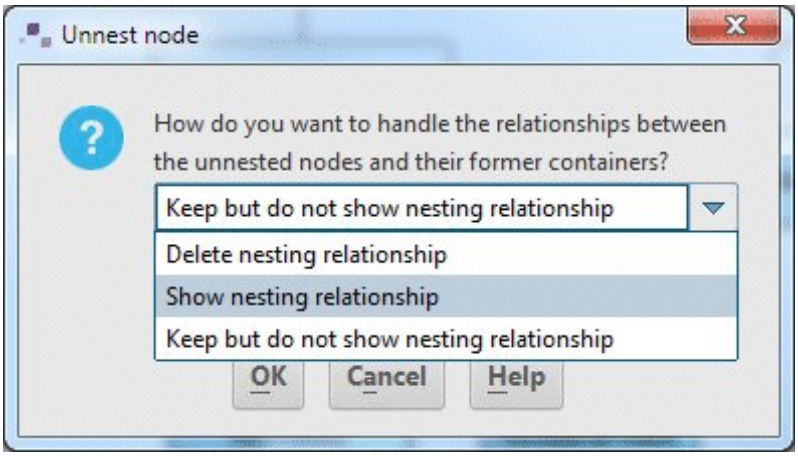


Figure 116: **Unnest node** dialog

Selecting **Delete nesting relationship** moves the class into the package in which the diagram is located. Selecting **Show nesting relationship** displays an Owning package relationship between the class and the package. The **Keep but do not show nesting relationship** option means that no relationship is displayed in the diagram; the class remains contained in the package at definition level.

A further option relates to the situation where one element can be nested in another in different ways.

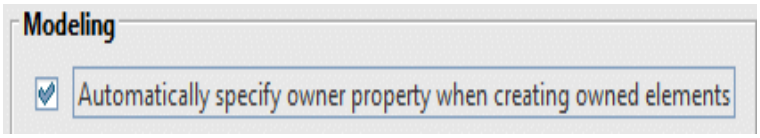


Figure 117: Option for creating a nested element

[26] See ARIS > Options > UML -> Designer – General > Nestings

In rare cases, a UML element can own another element of a particular type in more than one way without the semantic difference between the different types of ownership being obvious and really relevant to the user. One example of this is the UML type **Constraint**.

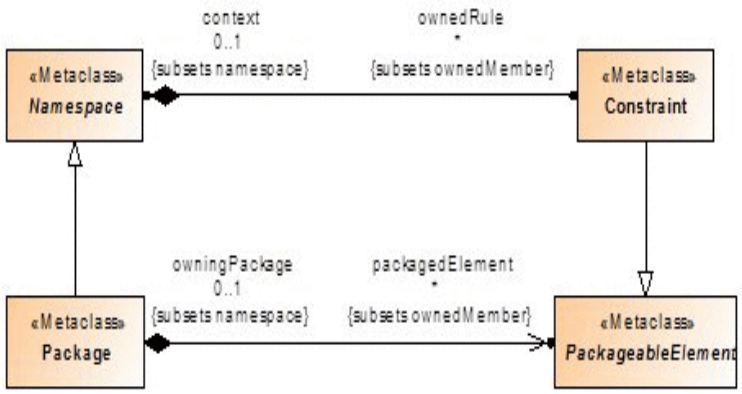


Figure 118: Extract from the UML metamodel with the Package and Constraint metaclasses

Every UML element whose type inherits from the **Namespace** metaclass can own constraints through its **ownedRule** property. Every UML package can own elements whose type inherits from the **PackagedElement** metaclass through its **packedElement** property. As the **Constraint** metaclass inherits from PackagedElement, you can either insert a constraint in a package as an ownedRule or as a packagedElement.

If the **Automatically specify owner property when creating owned elements** option is enabled, when creating a constraint in a package there is no query as to whether the constraint is to be contained in the package as an ownedRule or a packagedElement. In this case, it is automatically created as an ownedRule in the package.

You can also change the nesting type later by calling up the **Change nesting kind** item in the pop-up menu for the nested element. The **Select nesting type** dialog opens to select the corresponding nesting type.

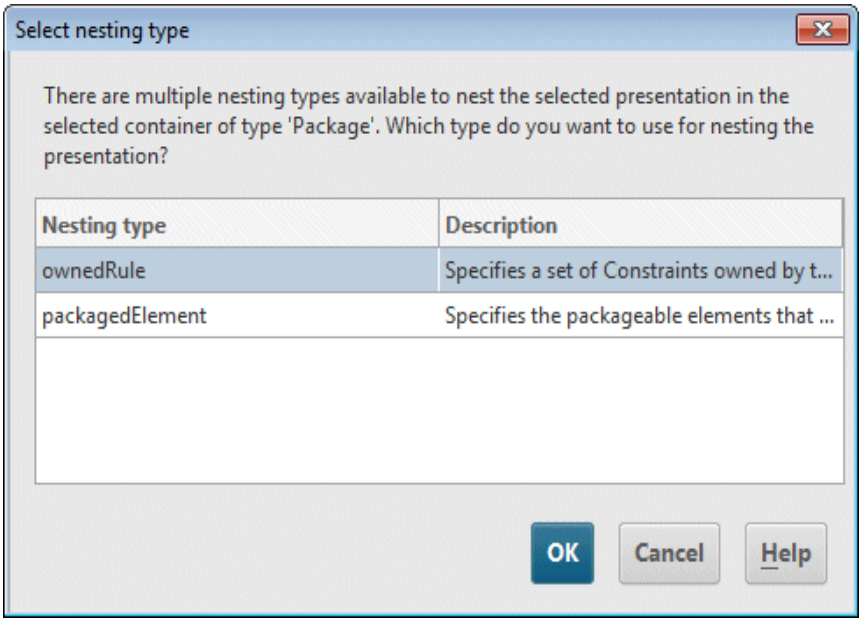


Figure 119: Dialog for changing the nesting type for a constraint contained in a package

3.3.4.7 Text nestings

Figure 120: Class with two attributes

Figure 120 shows the UML notation for a class with two attributes. The special feature of UML compared to the ARIS standard is that here a single presentation represents several elements and their relationships – a class (Item), two attributes (Item number and Description), a data type (String), two relationships of the **Class::ownedAttribute** type, and two relationships of the **TypedElement::type** type.

The two attributes are nested in the class presentation using text and, as such, can also be selected individually. The first click on the class selects the class itself (as shown in Figure 120). Clicking again on an attribute within the class selects the attribute. In this case the Properties area no longer shows the properties of the class but those of the attribute.

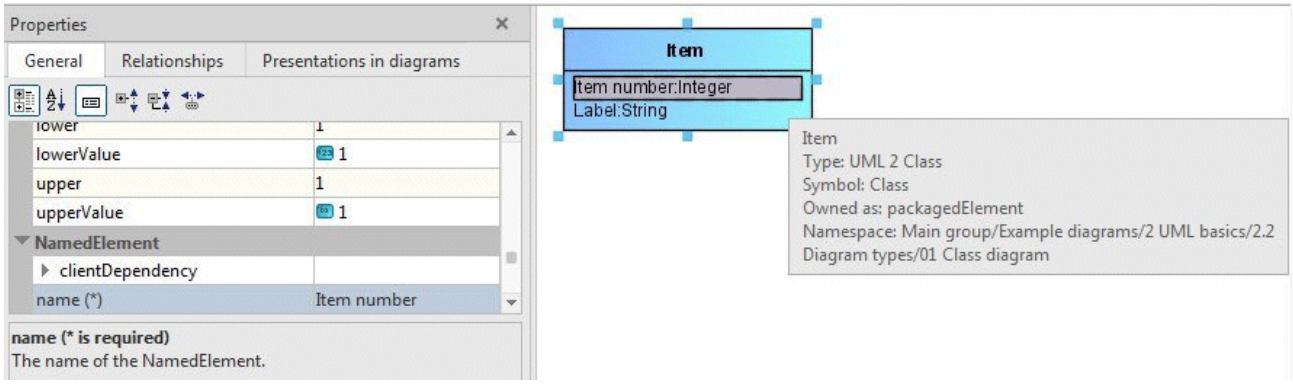


Figure 121: Class with selected attribute

Many of the functionalities that ARIS UML Designer provides for presentations are also available for textually nested elements. For example, you can move or copy them to another element, call up the Properties dialog and, last but not least, edit the text by clicking it a third time or pressing the **F2** key.

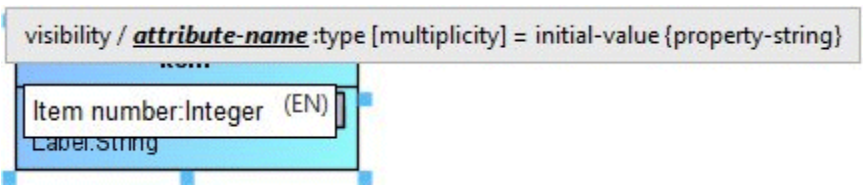


Figure 122: Text editing for an attribute in Designer

In Designer, you can also create new textually nested elements for an element using the pop-up menu:

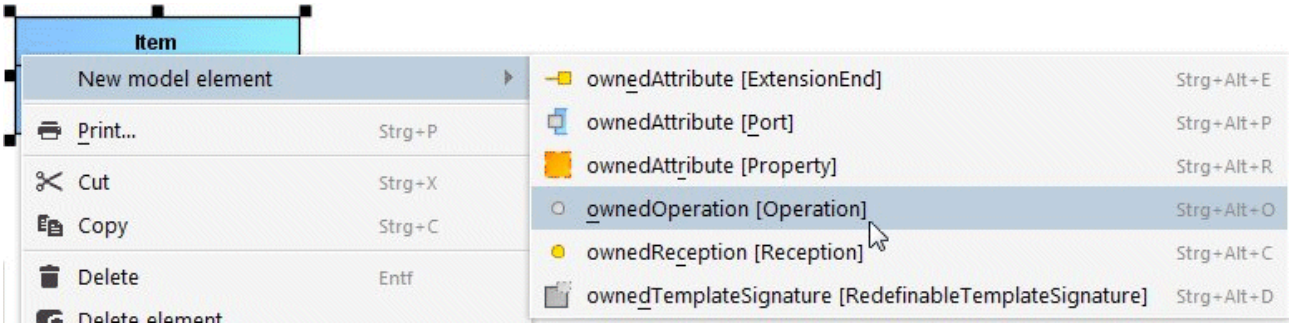


Figure 123: Creating a new operation using the pop-up menu

3.3.4.8 Modeling in groupings

ARIS UML Designer enables you to edit elements within groupings without having to cancel the grouping to do so.

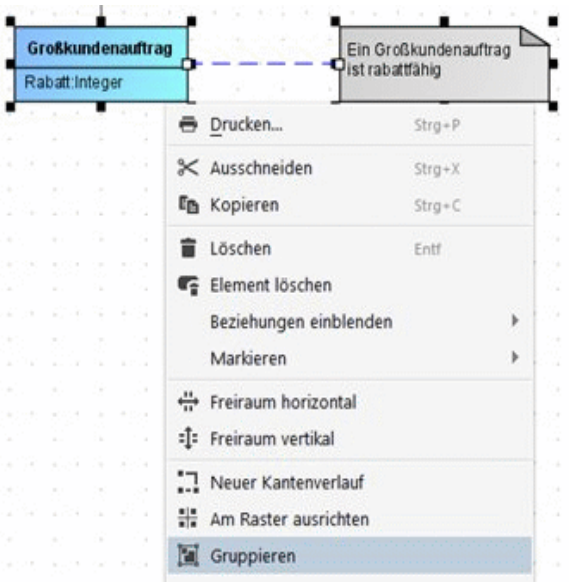


Figure 124: Creating a grouping

Groupings are created by selecting **Group** in the pop-up menu.

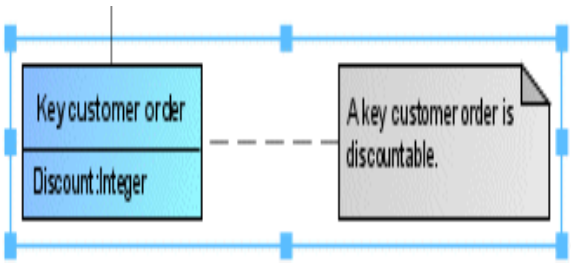


Figure 125: Grouped elements

You can move this kind of grouping in the diagram in its entirety without first having to select each element it contains.

However, you can still select individual elements within the grouping to edit their properties. If an element is in a grouping and the grouping is not yet selected, the grouping is selected the first time you click the element. Clicking the element again then selects the element itself, as shown in Figure 126.

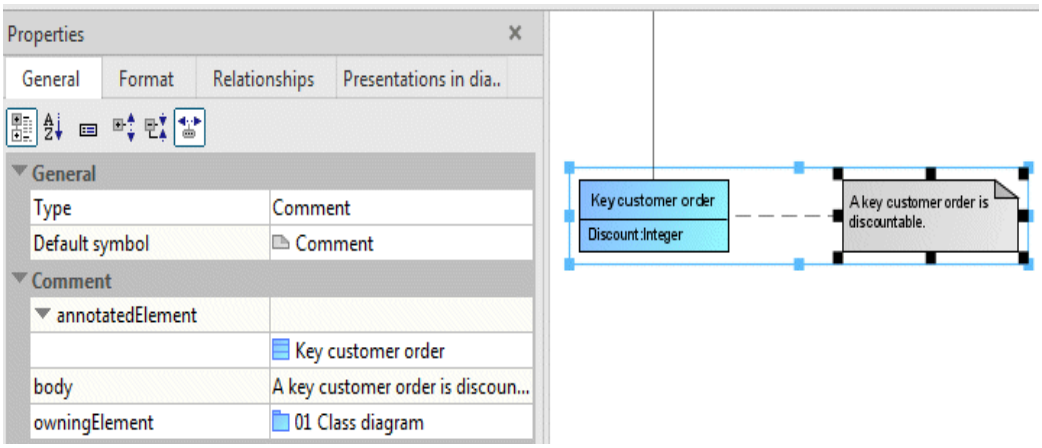


Figure 126: Selecting an element within a grouping

It is also possible to move elements within a grouping.

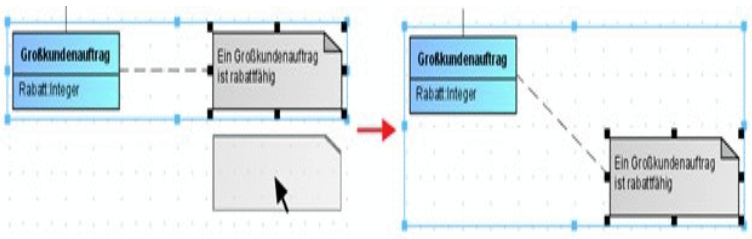


Figure 127: Moving an element within a grouping

3.3.4.9 UML-specific modeling support

A range of ARIS UML Designer functionalities are used to perform typical use cases in UML modeling, which normally require several manual editing steps, with just a few clicks. They include everything from simple use cases such as setting the multiplicity of association ends to default values using the pop-up menu through to more complex use cases such as creating port interfaces for components. A common feature of all functionalities is that they are available in the element's pop-up menu and under **Edit element** in the **Content** tab bar.

Two of these functionalities are introduced below by way of example.

3.3.4.9.1 Specifying the navigability of an association end

As described in section 2.4.4.2Associations, the navigability of an association end is not a simple Boolean property but depends on what the association end belongs to and, in some cases, the way in which the association end is assigned to its owner.

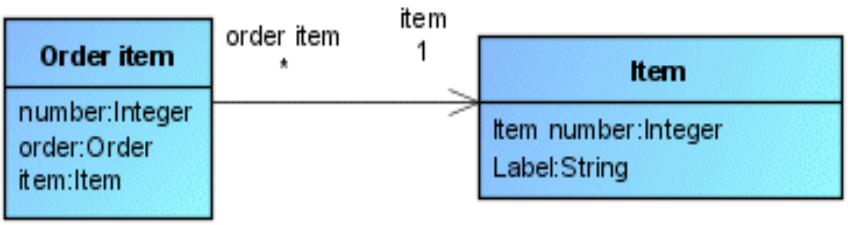


Figure 128: Unidirectional navigable association

Figure 128 shows two classes and an association. The **item** association end is navigable (indicated by the arrow head), as it is an attribute of the opposite class **Order item**. You can recognize this by the graphic representation with the dot at the arrow head, and also by the fact that it is listed in the class as an attribute[27].

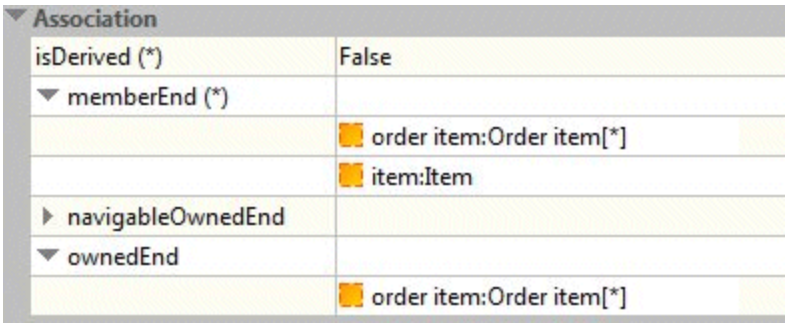


Figure 129: Association ends for the association

[27] Attributes that are simultaneously association ends are not normally also displayed in the attributes area of the class in the diagram. However, you can change this for the class by disabling the **Classifier > Hide association member ends** display option, which has been done here to illustrate the situation.

In the properties of the association, you can see that only one of the two association ends belongs to the association – the **order item** association end. As it is only listed for the **ownedEnd** property but not for the **navigableOwnedEnd** property, it is not navigable.

You can use the pop-up menu for the association end or the tab bar to easily change its navigability without having to manually make changes to the ownership relationships.

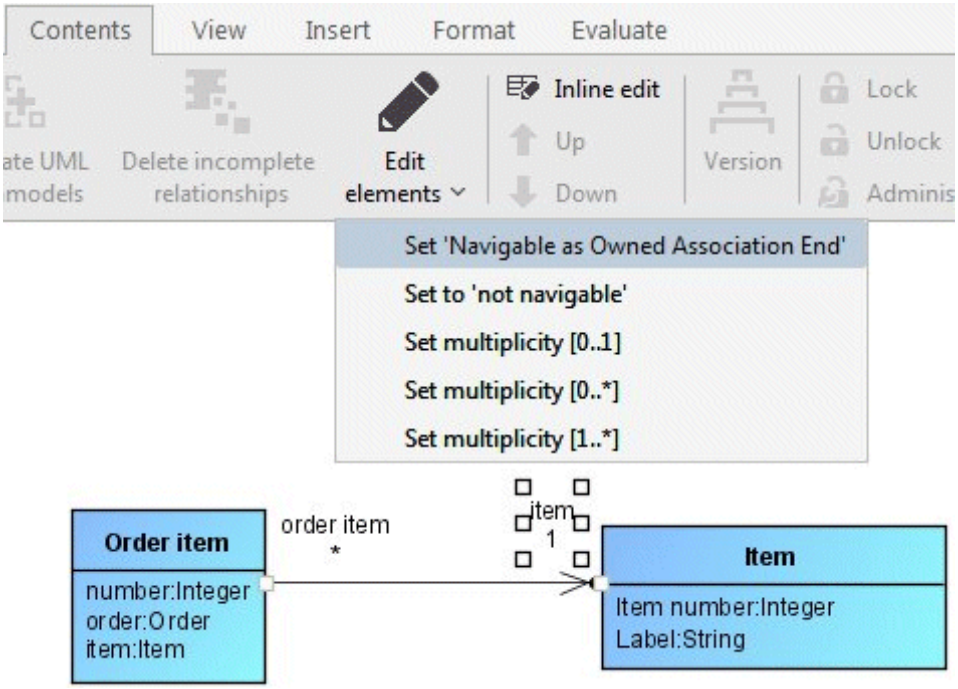


Figure 130: Functions for changing the navigability and the multiplicity

Figure 130 shows the UML-specific functionalities for an association end. The current status of the element is hidden in the list (**Set 'Navigable as Owned Association End'** and **Set multiplicity [1..1]**).

Selecting **Set 'Navigable as Owned Association End'** in the list displays the following screen:

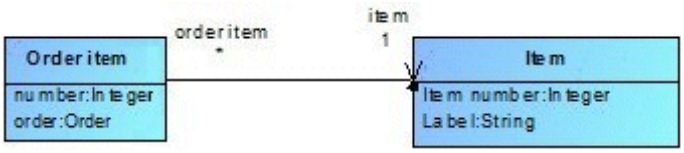


Figure 131: Association end navigable but not an attribute of the opposite class

The **item** association end is still navigable, but is no longer an attribute of the **Order item** class. This can also be seen from the changed properties of the association. The **item** association end is now listed for both the **ownedEnd** property and the **navigableOwnedEnd** property.

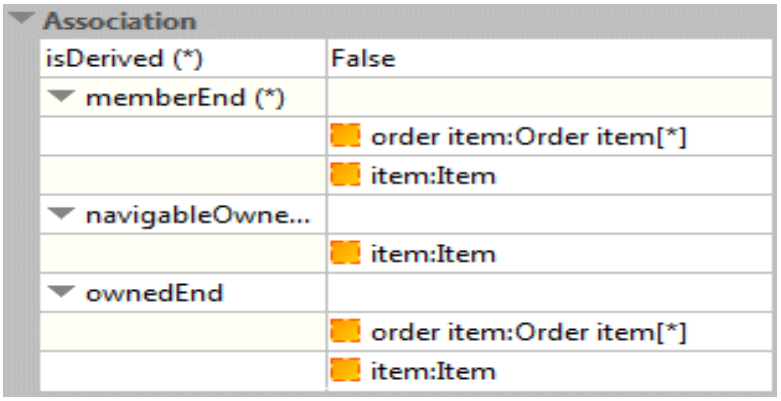


Figure 132: Association ends after changing the navigability

The functionalities for changing the navigability thus save the user having to manually edit the **ownedAttribute** property for the class at the opposite association end and the **ownedEnd** and **navigableOwnedEnd** properties for the association.

3.3.4.9.2 Creating getter and setter operations

The **Generate getter and setter** functionality provides an easy way to create corresponding access methods for the attributes of a class.

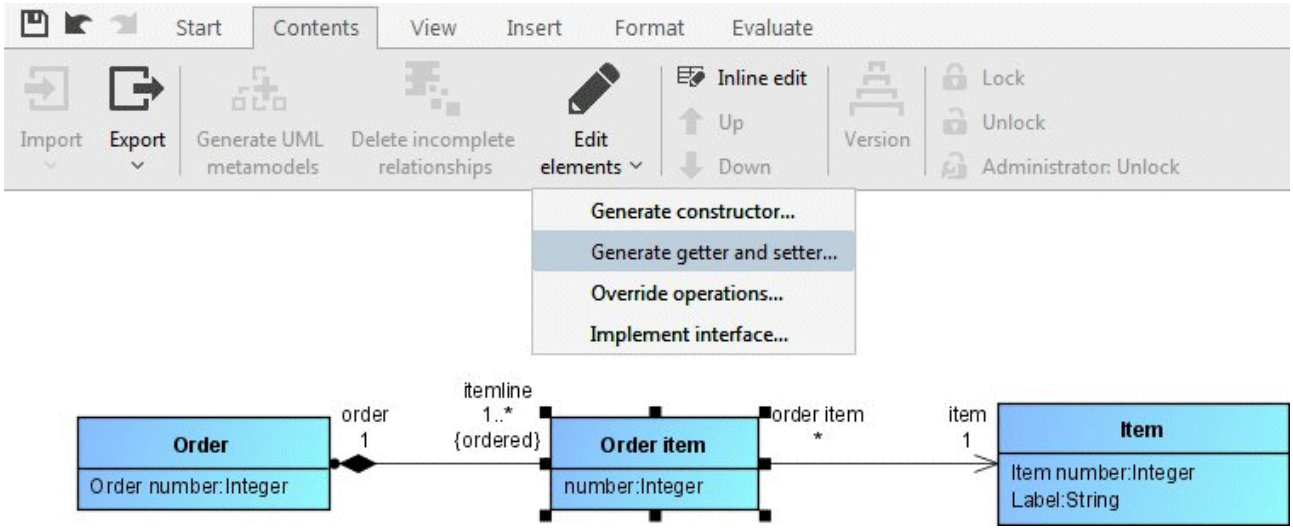


Figure 133: Functionalities for creating operations for a class

The functionality opens the **Generate getter and setter** dialog for selecting the attributes and setting some additional parameters for generating the access methods.

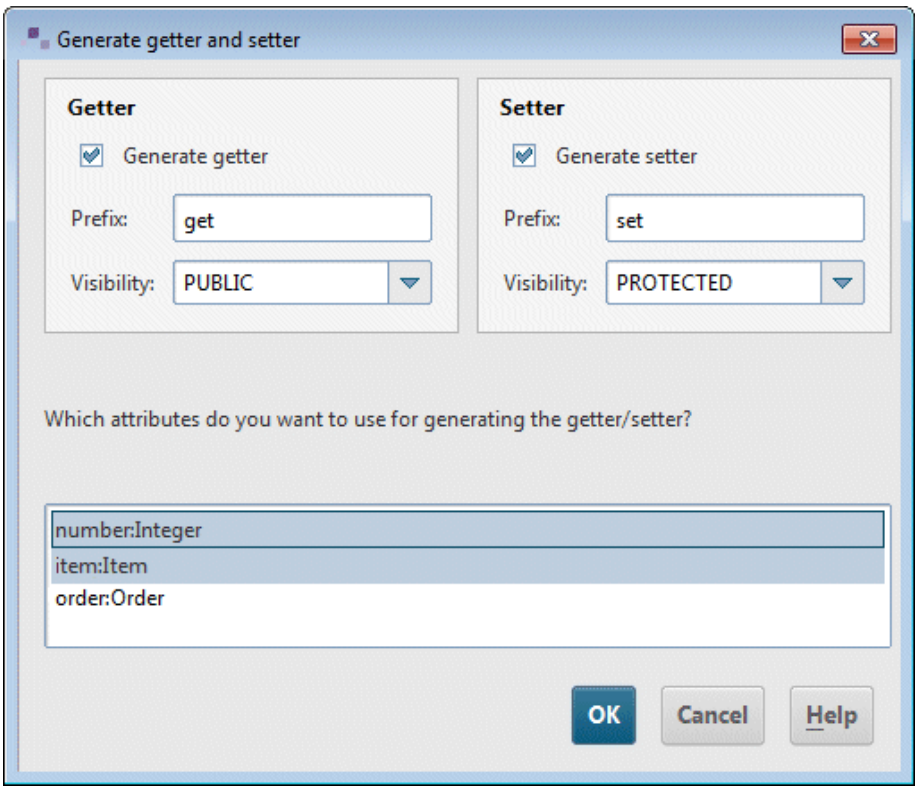


Figure 134: **Generate getter and setter** dialog

Figure 134 shows the **Generate getter and setter** dialog for the **Order item** class. The prefix **get** has been entered for the getters and **set** for the setters[28]. The two attributes **item** and **number** have been selected for generation. Clicking **OK** starts the generation process.

Figure 135 shows the class with the generated access methods.

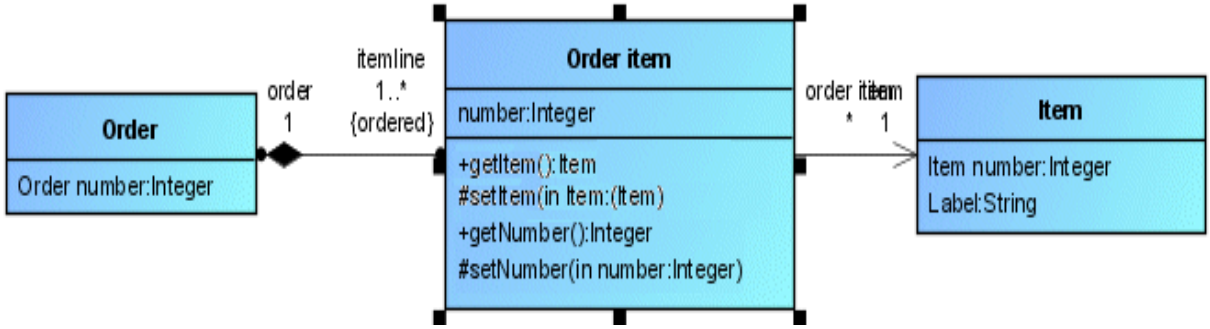


Figure 135: Generated getters and setters

[28] The two prefixes **get** and **set** appear by default when the dialog is opened.

3.4 Options

Selecting the **Options** menu item in the ARIS menu, you can open the dialog for editing the general settings for ARIS UML Designer.

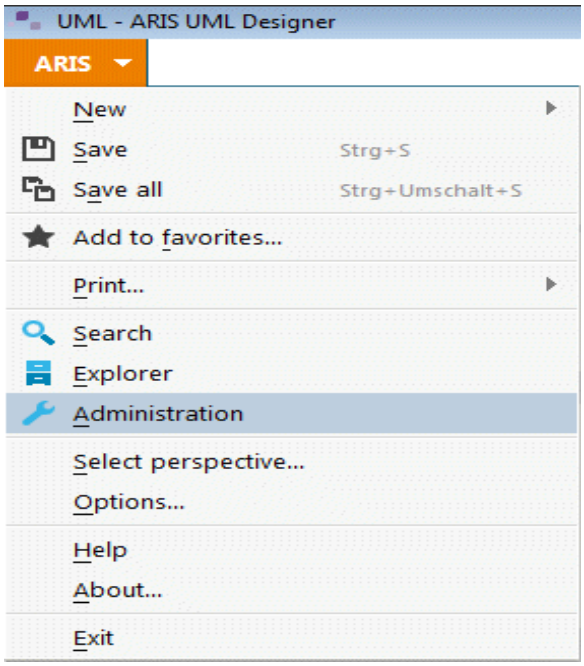


Figure 136: Menu item for editing the options

The dialog contains a series of options, the most important of which are outlined below. The options for the **Perspective**, **Print**, and **Versioning** topics are not discussed here, as they are ARIS standard options.

Changing some options, e.g., language options or working environment, require you to restart ARIS UML Designer.

3.4.1 General

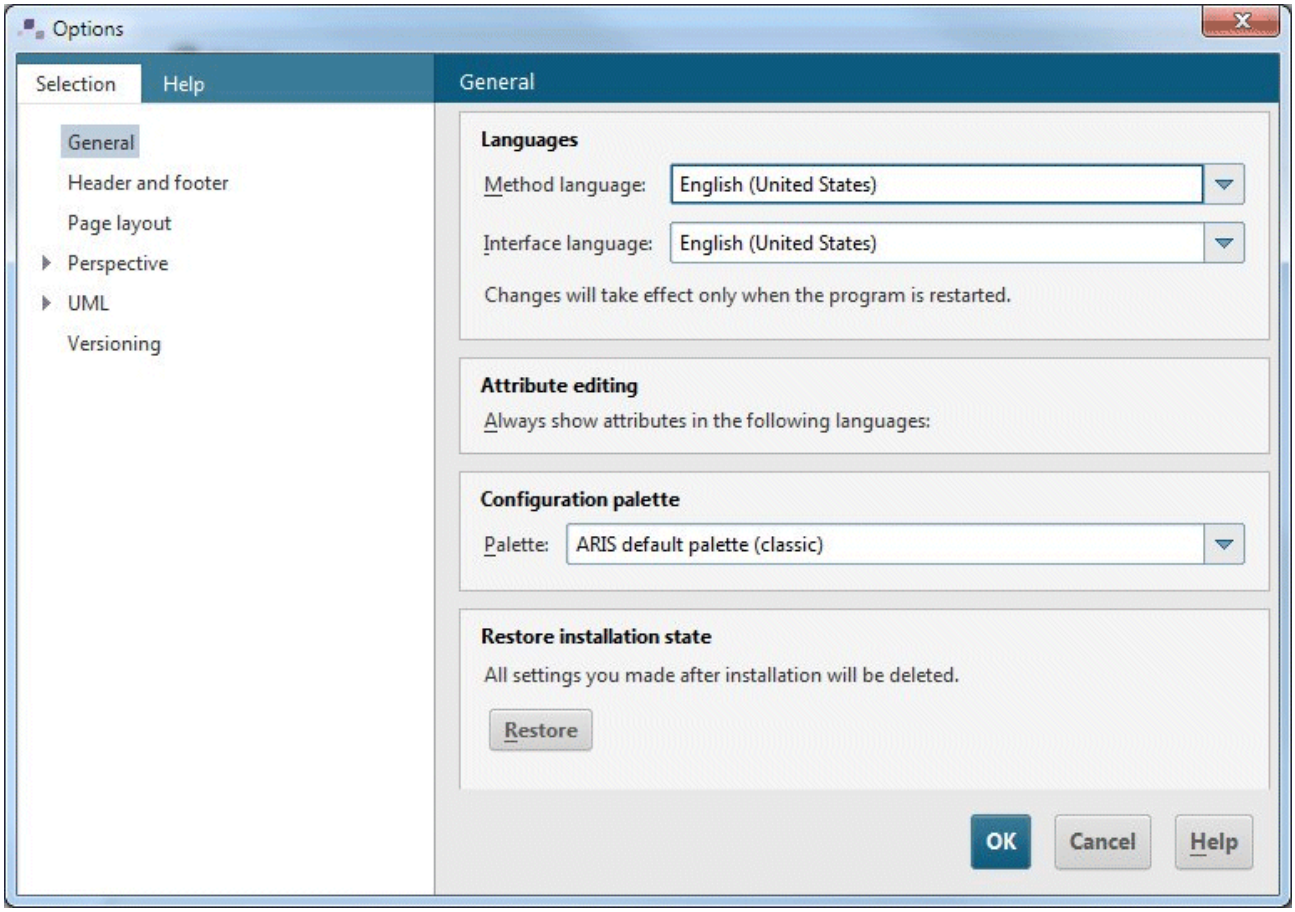


Figure 137: **General** options page

The Method language specifies the language in which the names of diagram types, element types, and properties that do not relate to UML are displayed. There is a separate language setting for UML type names in the UML-specific options.

The Interface language specifies the language in which texts in the user interface are displayed.

In ARIS UML Designer, the palette only affects the display of the symbols for non-UML elements in the Explorer tree.

3.4.2 UML - General

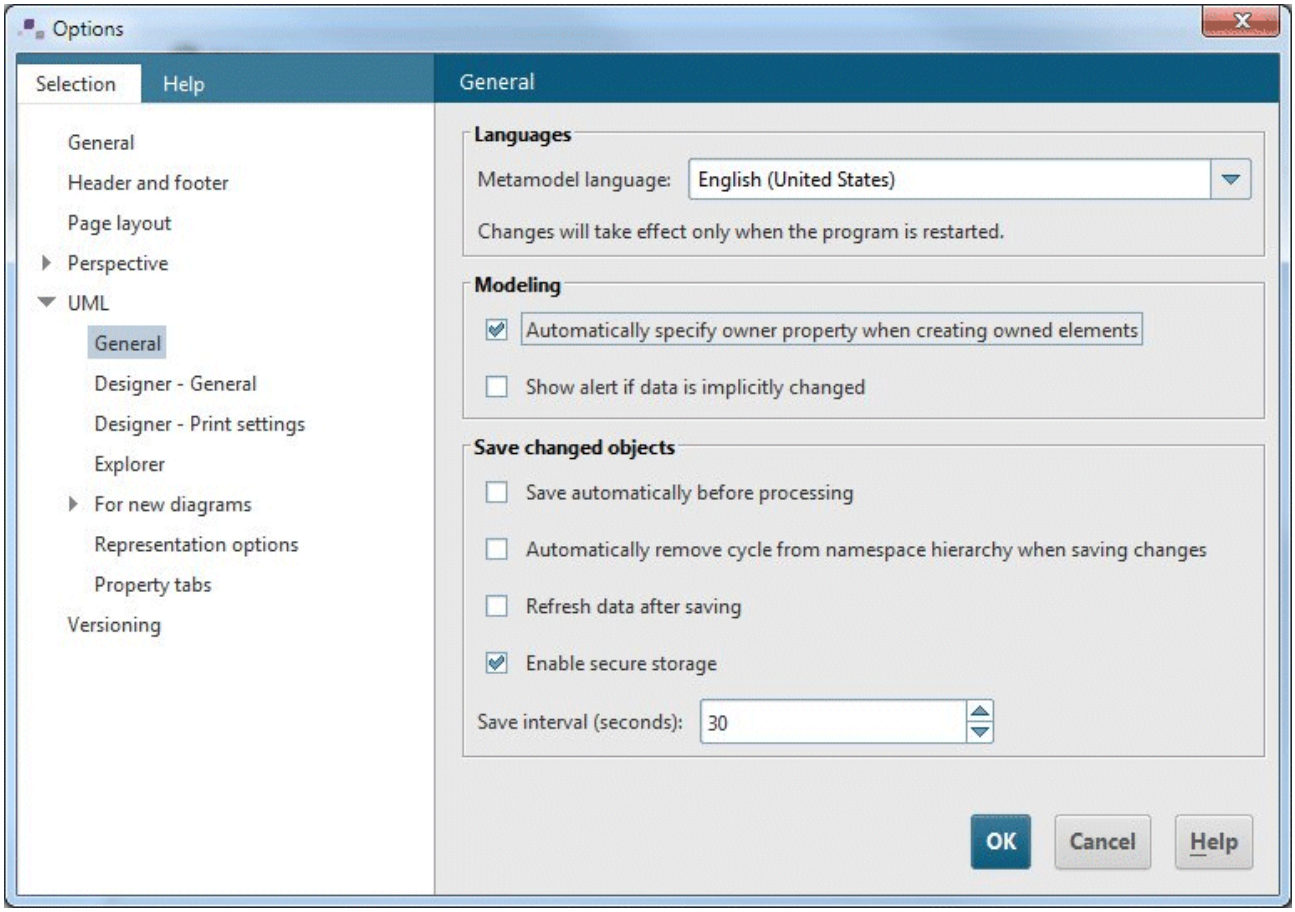


Figure 138: **UML > General** options page

The Metamodel language specifies the language in which the type names of UML elements, UML properties, and UML diagrams are displayed. For example, this allows object types, diagram types, and attribute names for non-UML content to be displayed in German, and UML types in English.

The option for selecting the owner property has already been explained in section 3.3.4.6Graphic nestings.

The **Show alert if data is implicitly changed** option can be used to enable and disable the logging of implicit changes described in section 0Implicit changes bar.

Some changing functionalities of ARIS UML Designer are executed on the server as they potentially involve large data volumes. They require all changes made in the client to be saved first. If the **Save automatically before processing** check box is enabled, saving occurs with no confirmation when the corresponding functionality is called. Otherwise, a dialog appears asking whether you want to save or cancel the functionality.

When multiple users are working on the same structures in a database, it is theoretically possible that two users could simultaneously change the structure in such a way that the two changes would combine to cause a cycle in the element hierarchy. As soon as one user has saved his changes, in this situation the second user's save will fail if the cycle is not fixed first. If the **Automatically remove cycle from namespace hierarchy when saving changes** check box is enabled, the fix is carried out when saving without confirmation. Otherwise a corresponding dialog appears.

Figure 139: Dialog when saving with a cycle in the namespace hierarchy

If the **Refresh data after saving** check box is enabled, all data is reloaded from the server after saving, so that changes made by other users are included.

The **Enable secure storage** check box is used to set ARIS UML Designer to back up all changes locally at regular intervals. You use the **Save interval (seconds)** setting to specify how often this is to be performed. If the connection to the server is lost due to network problems, at the next login you will be asked whether you want to restore the changes that are not saved in the database.

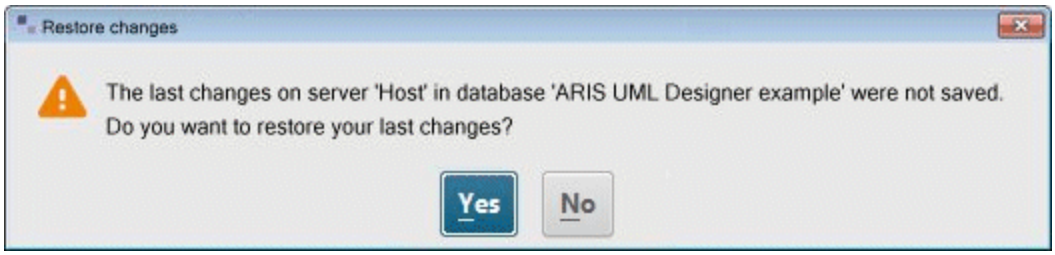


Figure 140: Confirmation prompt for restoring unsaved changes

3.4.3 UML - Designer-General

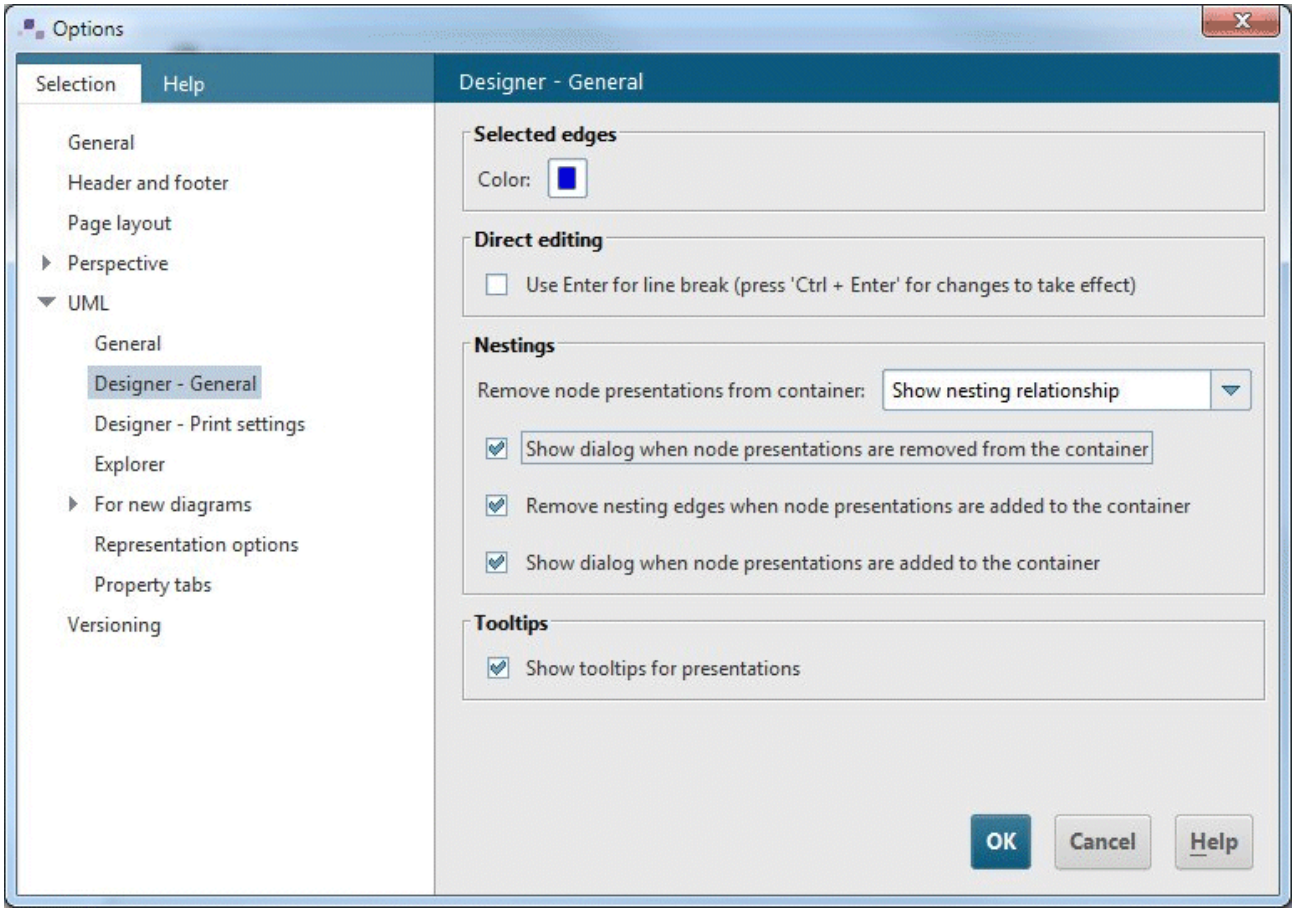


Figure 141: **UML - Designer > General** options page

The **Selected edges - Color** option specifies the color in which edges selected in diagrams are displayed.

Changes in text input boxes are normally applied using the Enter key, and line breaks are entered using **Ctrl + Enter**. You can reverse this by enabling the **Use Enter for line break (press 'Ctrl + Enter' for changes to take effect)** check box.

The options for nestings have already been explained in section 3.3.4.6Graphic nestings.

If the **Show tooltips for presentations** check box is enabled, a tooltip showing information about UML elements in diagrams is displayed at the mouse position.

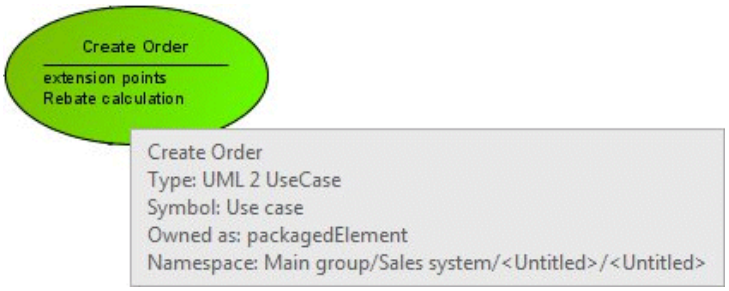


Figure 142: Tooltip for a UML element in a diagram

3.4.4 UML - Explorer

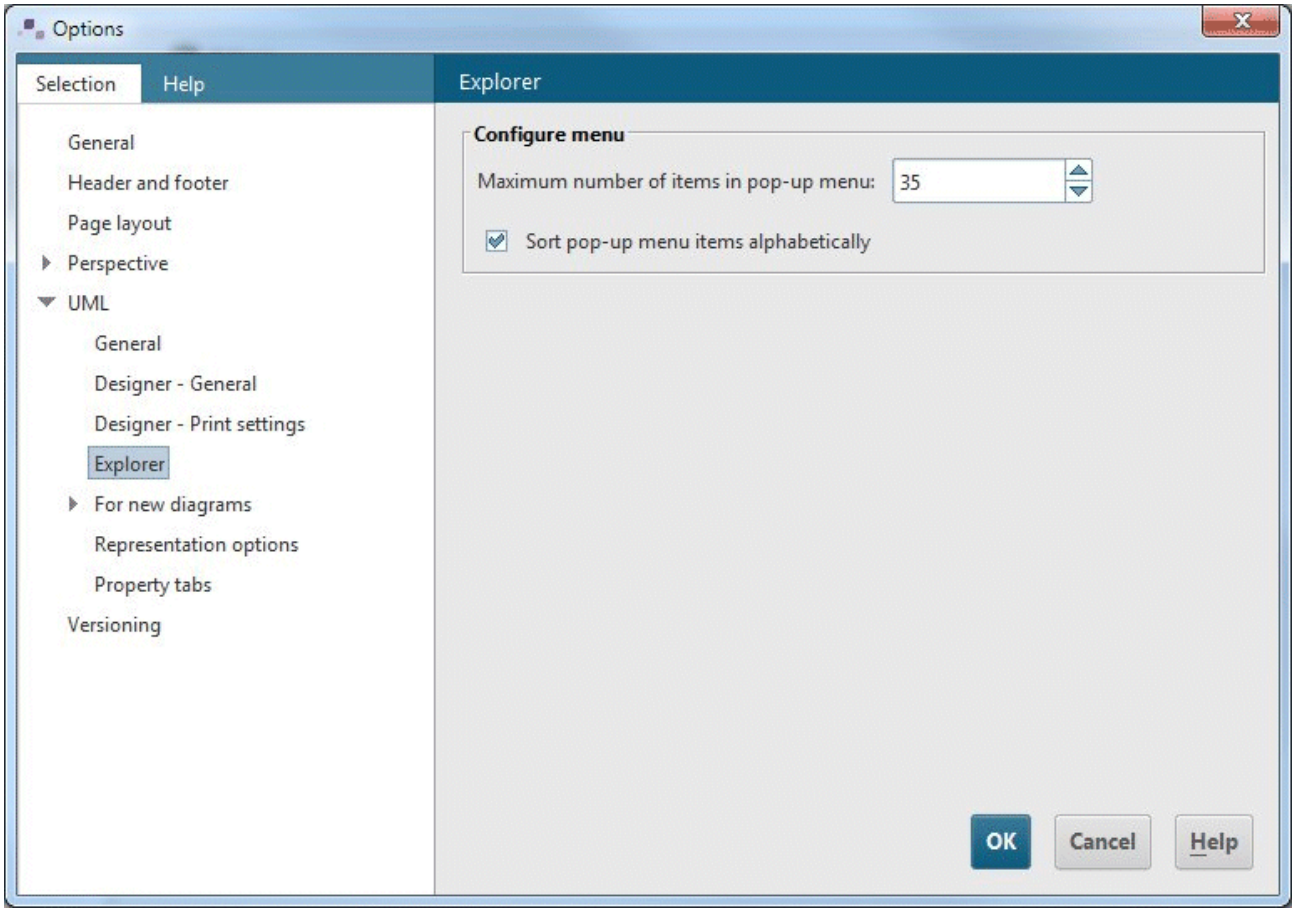


Figure 143: **Explorer** options page

Here, you can set how many items the Explorer pop-up menu for creating new elements displays, and whether the items are to be sorted alphabetically (see section 0Creating new elements in Explorer).

3.4.5 UML - For new diagrams

These option pages relate to general representation options for diagrams, which are also supported in the ARIS standard. Only the following option here is specific to ARIS UML Designer:

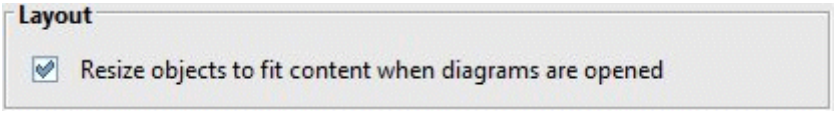


Figure 144: Layout option on the UML - For new diagrams - Representation options page

Enabling the check box means that when a diagram is opened the size of the elements is adjusted to their content.

The settings made on these options pages only affect new diagrams you create. Existing diagrams are not changed.

3.4.6 UML - Representation options

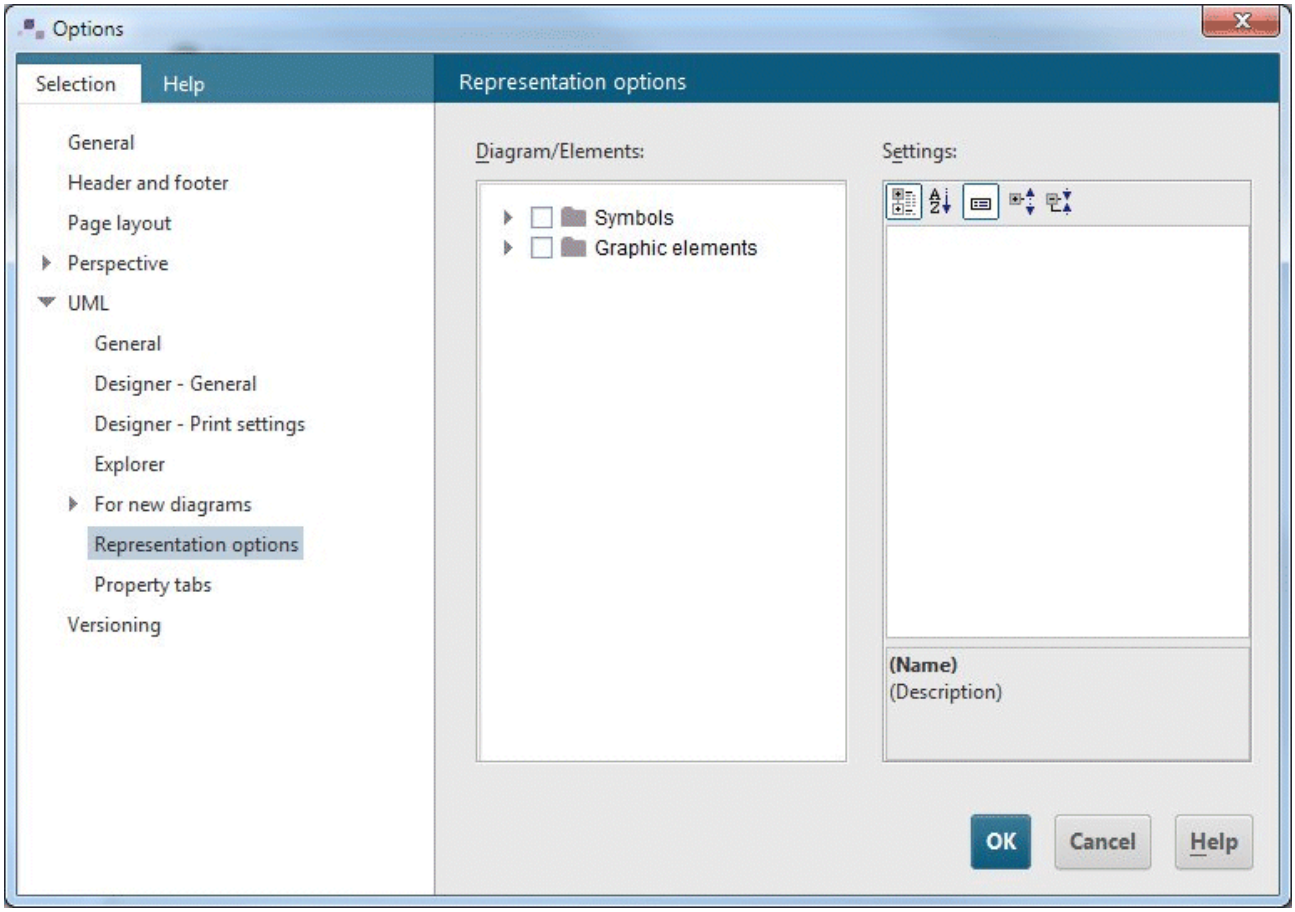


Figure 145: **UML - Representation options** options page

On this options page you can specify the default settings for the representation options for all UML symbols and graphic elements. The symbols are grouped by diagram types. The options can be individually specified for specific symbols by selecting just one symbol and editing its representation options.

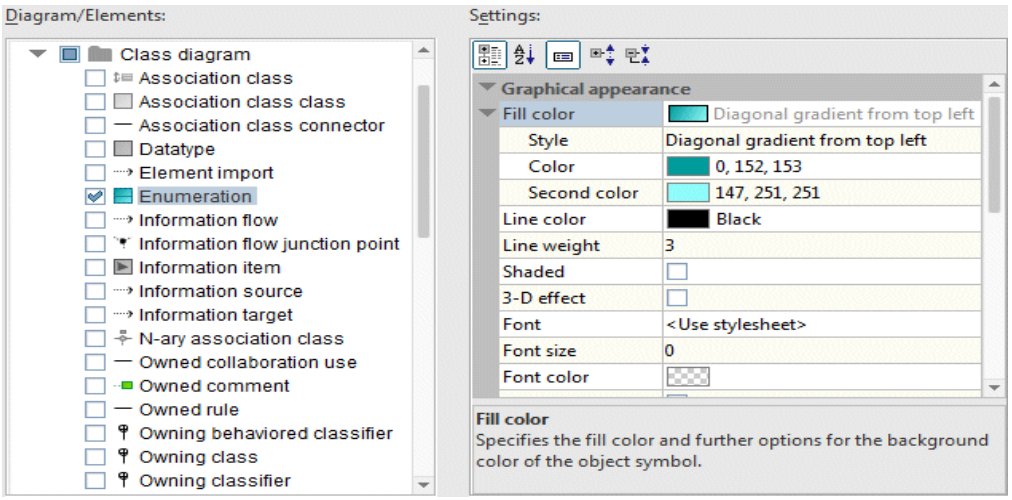


Figure 146: Selecting the **Enumeration** symbol

You can also select multiple symbols or entire hierarchy levels. In the example below, the **Shaded** and **3-D effect** properties have been disabled for all UML symbols.

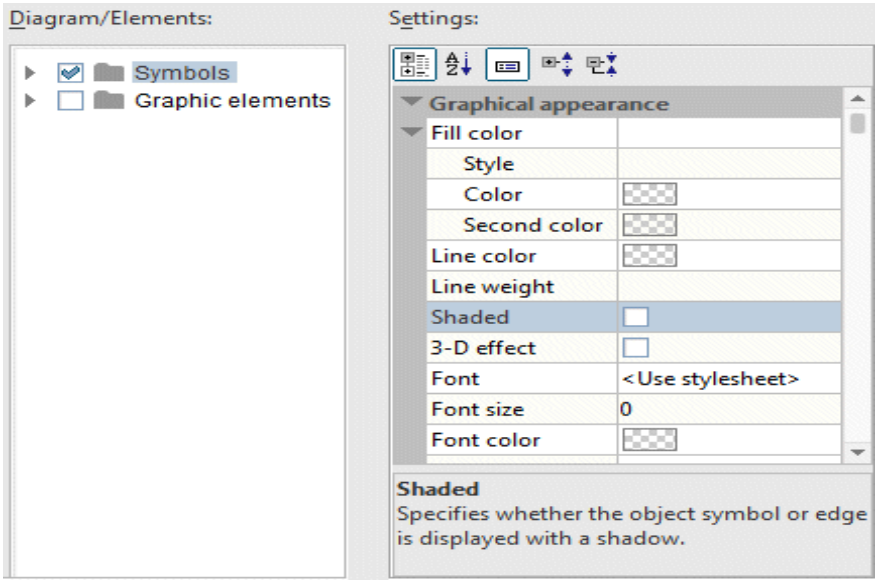


Figure 147: Selecting all symbols by selecting the top level

The changes made on this options page only affect new presentation elements you create. Existing elements are not changed.

3.4.7 UML - Property tabs

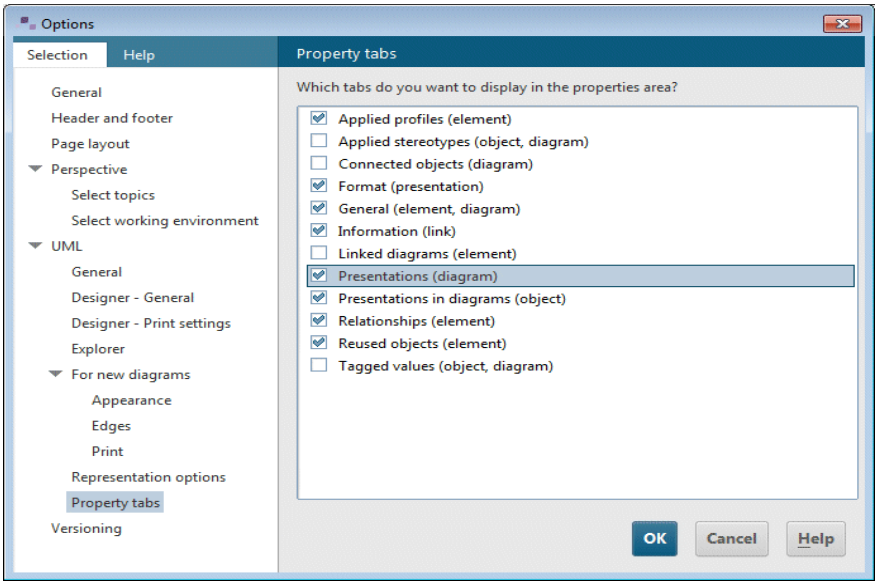


Figure 148: **UML - Property tabs** options page

Here, you can select which property tabs are to be displayed in the Properties area of the Designer component.

These options have no effects on the **Explorer** tab. It always shows all property tabs as considerably more space is available there.

3.5 Administration tab

The **Administration** tab provides various administrative functionalities. These include configuration of the method filters, management of access privileges, and writing of reports. There are also some functionalities specific to UML Designer, such as configuration of the link between business process and UML modeling, and management of XMI resources.

The **Administration** tab is only available if you have selected the **Configuration & Administration** working environment (see section 3.1Specifying the working environment). It is opened by selecting the **Administration** menu item in the ARIS menu.

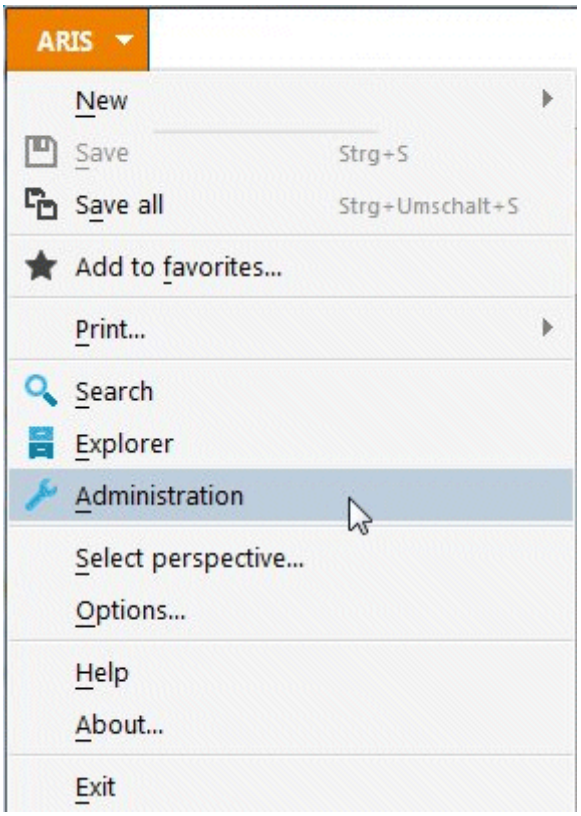


Figure 149: Menu item for starting Administration

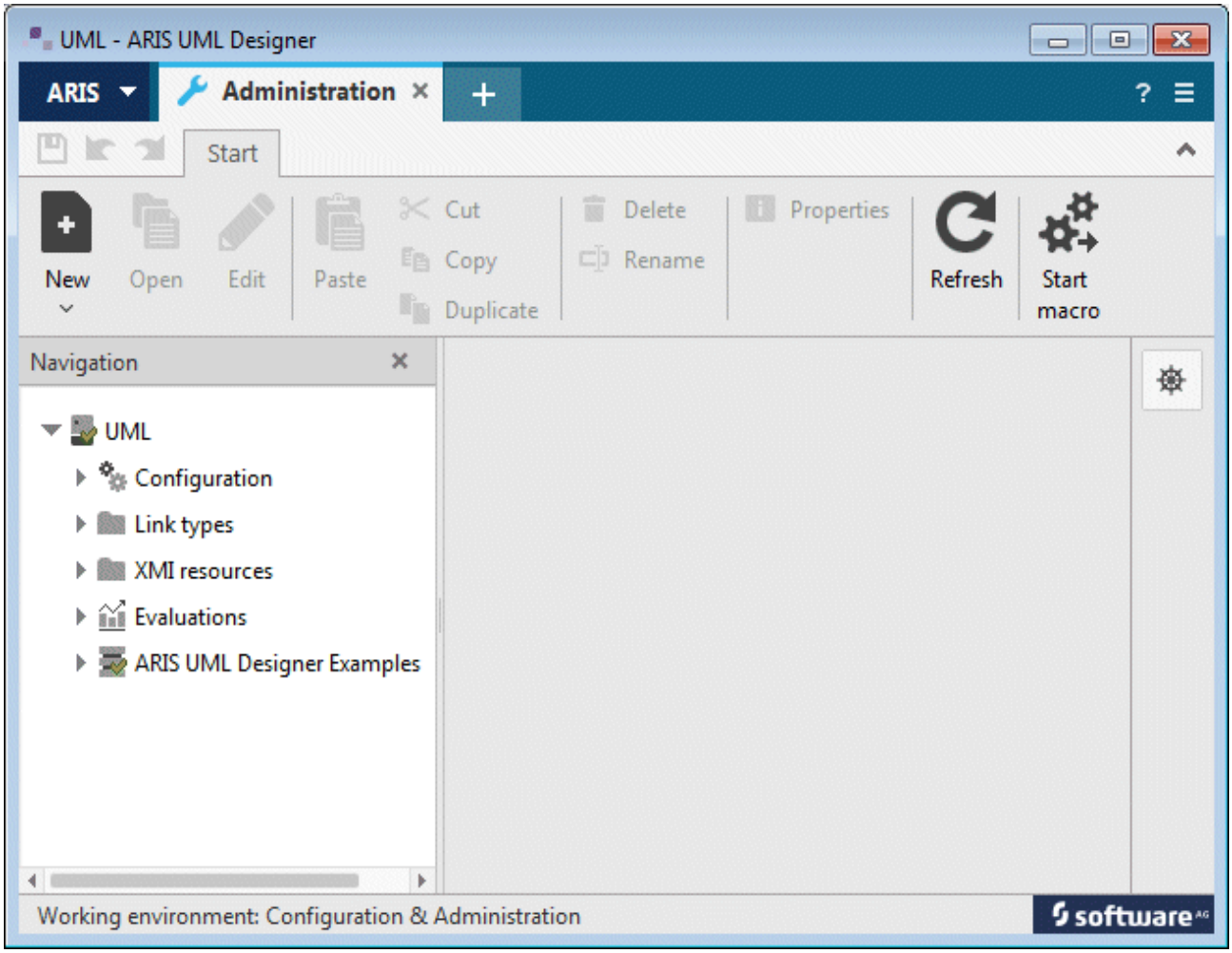


Figure 150: Administration component

Only those aspects of the **Administration** tab that are specific to ARIS UML Designer are discussed below.

3.5.1 Method filter

When you log into a database in ARIS UML Designer, only those method filters that include UML 2 are available for selection. Unlike in the ARIS standard, UML 2 can only be contained in the method filter in its entirety. A user-defined extension of the UML method is not possible. This is because of the high complexity of the UML metamodel and the fact that the UML specification for user-defined extensions and restrictions includes the use of UML profiles. An introduction to the topic of UML profiles can be found in section 6UML profiles.

Below, the default filter is used to demonstrate how you can add UML to the method filter.

Select the **Default** filter in the filter list and select **Edit** from the pop-up menu or the tab bar.

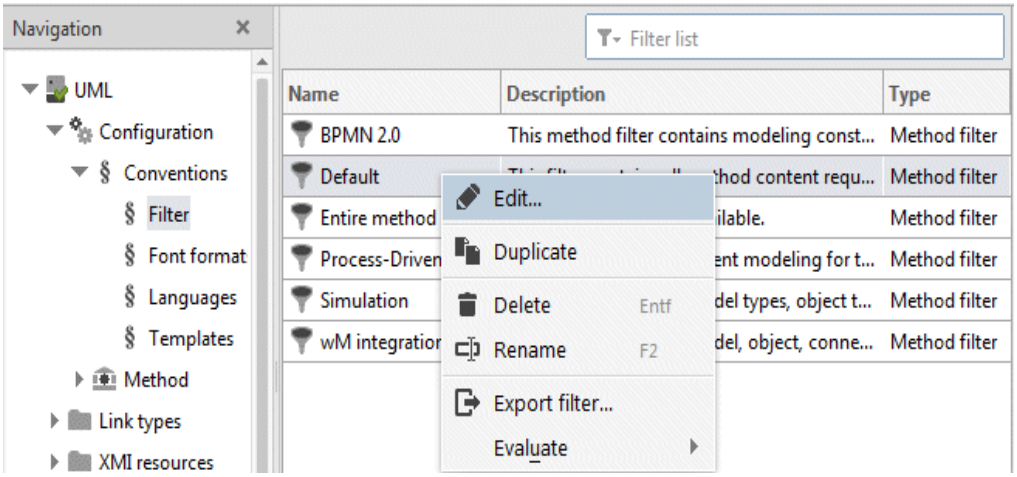


Figure 151: Editing the filter

In the Filter Wizard, click the **Next** button to navigate to the third page **Select metamodels** and enable the **UML 2.5** option. Then click **Finish** to extend the filter with UML 2.

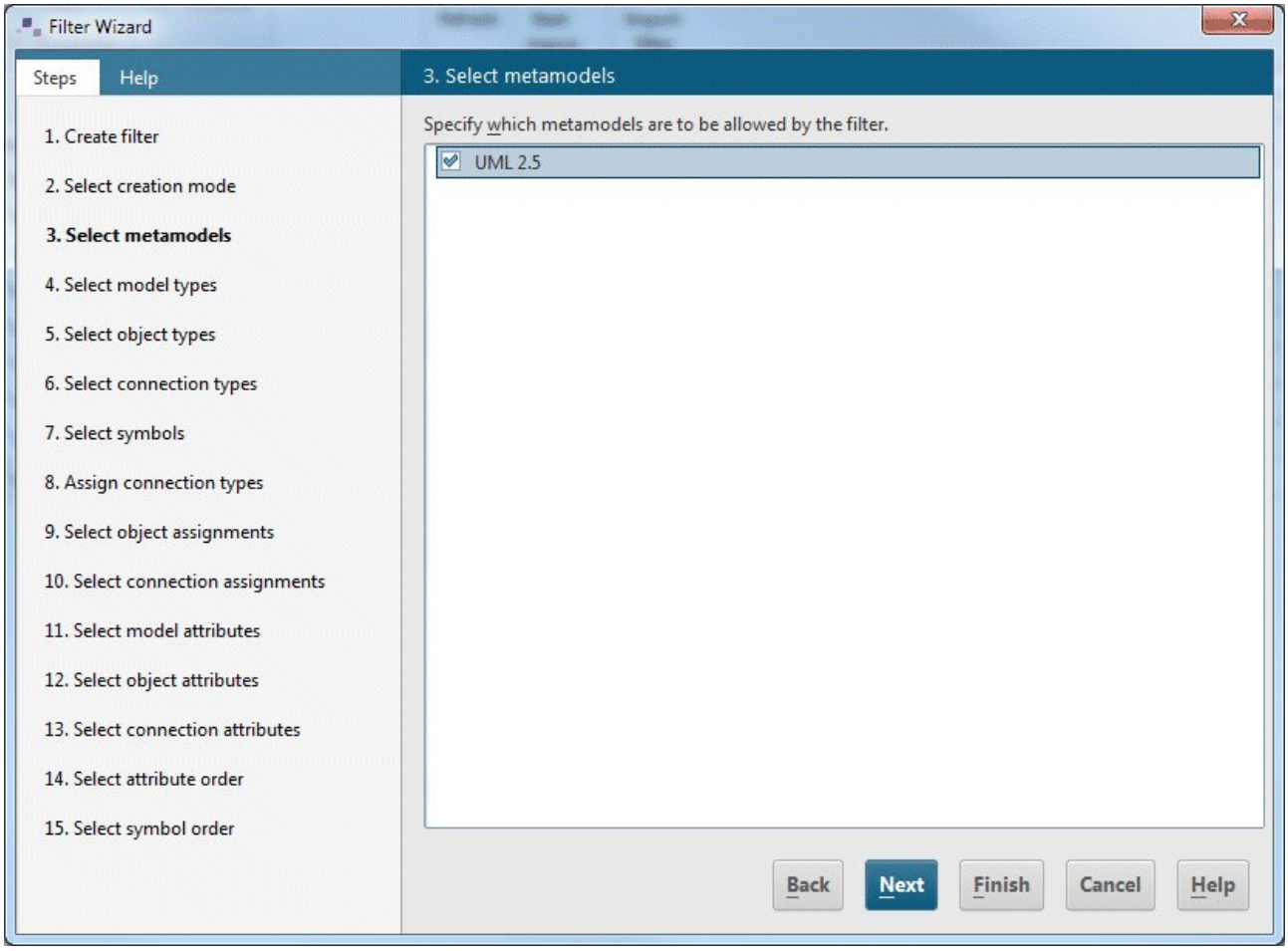


Figure 152: Selecting the UML 2.5 metamodel in the Filter Wizard

3.5.2 Link types

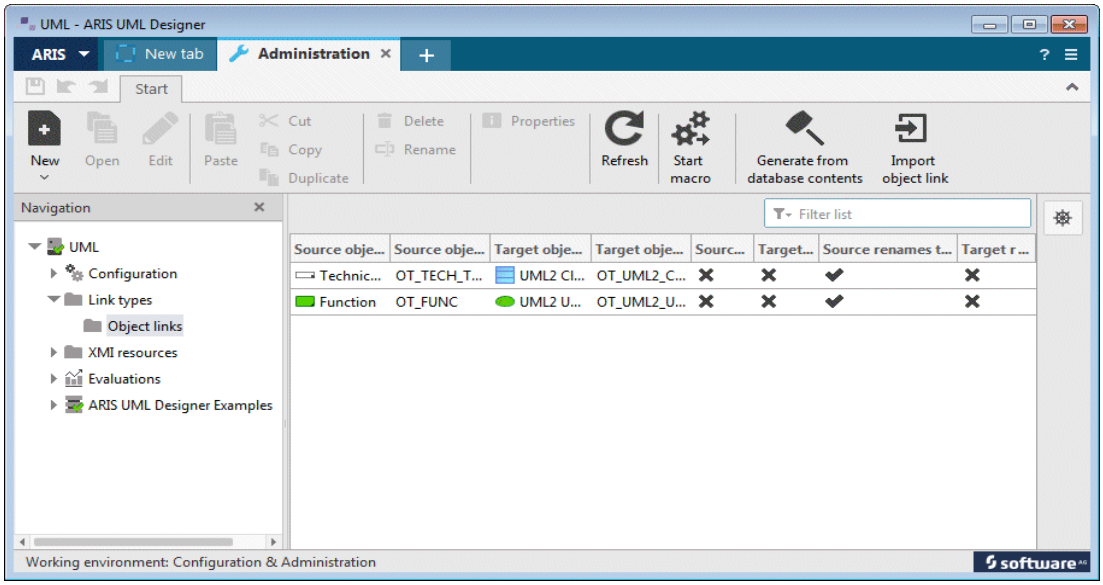


Figure 153: Managing the link types

Here, you can define which business process objects you want to map to UML and specify rules for the mapping. This functionality is outlined in detail in section 5Linking business process and UML modeling.

3.5.3 XMI resources

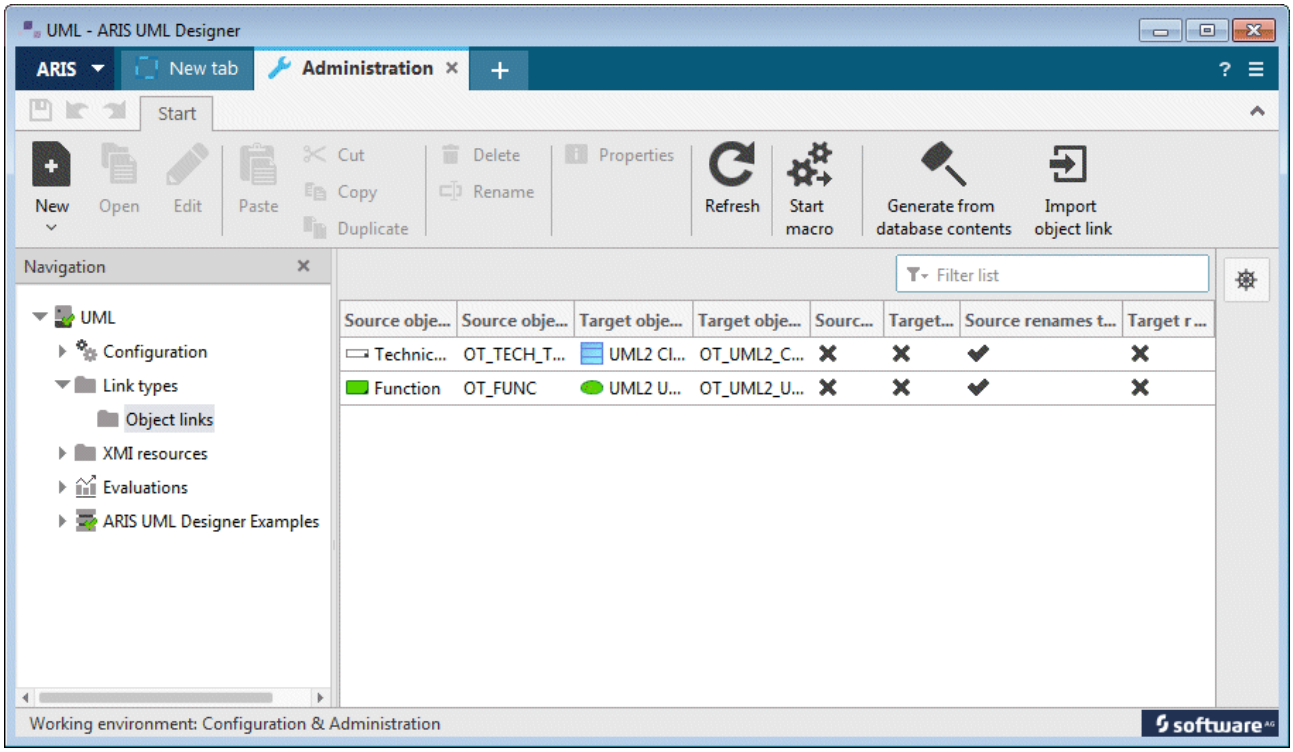


Figure 154: Managing XMI resources

XMI (XML Metadata Interchange) is a format for exchanging metamodel-based data between different tools. Like UML, XMI is a standard defined by the OMG.

ARIS UML Designer exports and imports XMI files in UML 2.5 / XMI 2.1 format. You can use XSLT files to make corresponding adjustments to XML formats from third-party manufacturers. These XSLT files are managed in the **XMI resources** area and can be selected as options during the XMI export and import.

The topic of the XMI interface and XSLT files for adjustment to external formats is discussed in more detail in the document **ARIS UML Designer XMI interface**.

3.5.4 Data transfers from ARIS UML Designer 7.x

If a database contains UML content from ARIS 7.x, which was created using ARIS UML Designer 7.x, it must first be converted to UML 2 before it can be displayed or edited with ARIS UML Designer 9.

A detailed description of UML conversion can be found in the document **UML Migration Guidelines**.

4 Mapping from UML to the ARIS object model

UML is completely mapped to the ARIS object model. However, two crucial aspects of UML required an extension of the ARIS object model compared to ARIS 7:

- UML elements can contain other UML elements and UML diagrams, they can occur in diagrams, and they can be linked to one another by relationships.
- A graphic edge in a UML diagram can represent an entire series of UML elements and relationships.

These aspects are discussed in more detail below.

4.1 Group and object properties of UML elements

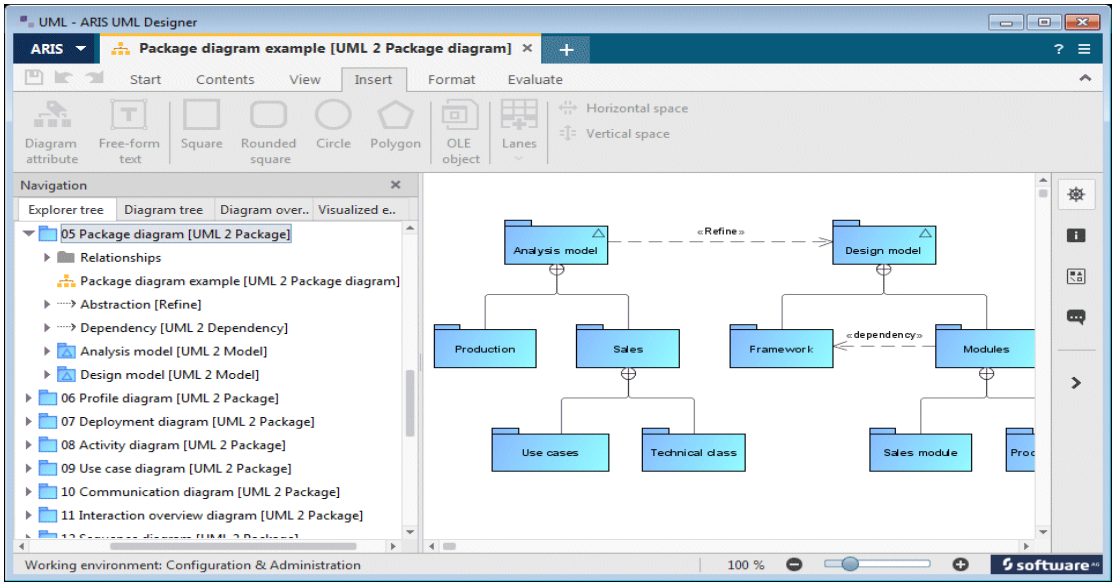


Figure 155: Package hierarchy and Package diagram

Figure 155 shows a hierarchy of packages with a diagram in the Explorer tree on the left, and a package diagram with packages and their relationships with one another on the right.

Both sides show aspects typical of various ARIS types for packages.

The package hierarchy shows typical features of an ARIS group hierarchy. Just as ARIS groups can contain other ARIS groups, ARIS objects, and ARIS models, UML packages can also contain other UML packages and UML diagrams. Therefore, it would be obvious to map UML packages to ARIS groups.

By contrast, the diagram shows typical features of ARIS objects. Just as ARIS objects can be contained in ARIS models as object occurrences and can be linked to one another by connections, UML packages are contained in the diagram as element presentations and are linked to one another by edges. Looking at this aspect alone, it would be obvious to map UML packages to ARIS objects.

This contradiction in the mapping of UML to ARIS has been resolved by assigning the ARIS type **Group** all properties of the ARIS type **Object**. This means that groups have an object type, they can have occurrences in diagrams, and they can be linked to one another by connections.

As every UML element can ultimately contain other UML elements (every UML element can own elements of the UML type **Comment**) and many elements can also own diagrams, UML elements are mapped to groups of the relevant UML type in ARIS.

For example, this means that a use case (UML type **UseCase**) is saved in ARIS as a group with the object type **OT_UML2_USE_CASE**. This mapping applies to all UML elements, regardless of their appearance in diagrams. A generalization (UML type **Generalization**), shown graphically as an edge in diagrams, is also saved in ARIS as a group with the object type **OT_UML2_GENERALIZATION**.

The exceptions are certain elements that normally appear right at the bottom of the element hierarchy and, at the same time, occur frequently. Examples of these elements are LiteralInteger and LiteralUnlimitedNatural. They are normally used as the lower or upper limit for multiplicities on association ends and attributes.

To ensure that management of user privileges does not become too fine granular, the possibility of defining user privileges has been restricted to UML elements of the **Package**, **Model**, and **Profile** types.

The behavior of conventional groups has not changed in ARIS. They have the object type **Group** (OT_GROUP). For groups of this type, there is still no facility for them to occur in diagrams or to be linked by connections.

4.2 Complexity of edge presentations

Figure 155 also shows the second aspect mentioned, namely the fact that graphic edges in diagrams represent both direct relationships between the packages – the edges between packages and their contained packages – and also other UML elements that are used to link the packages to one another, e.g., the Dependency relationship.

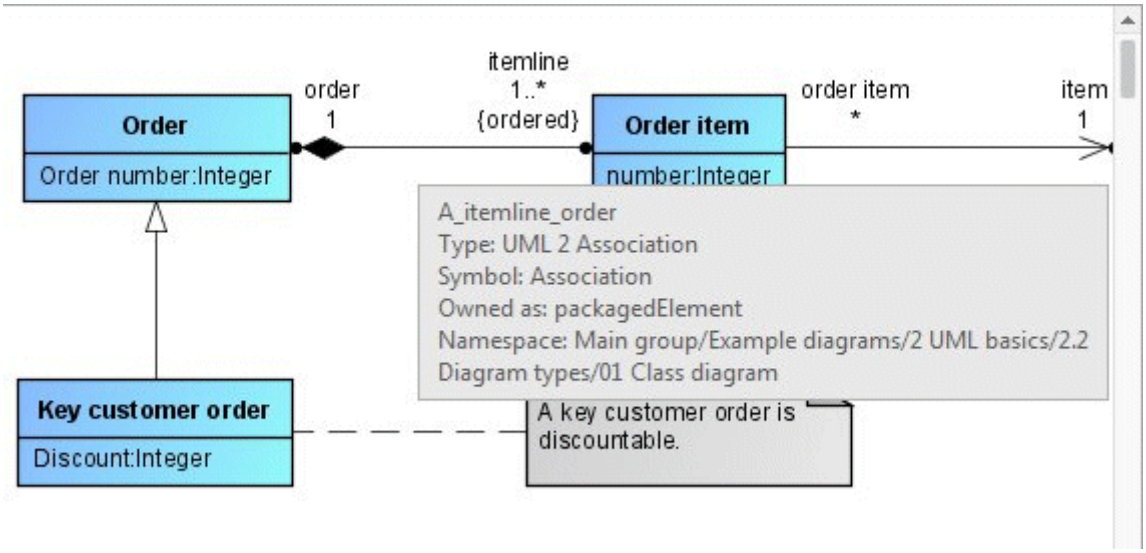


Figure 156: Association as graphic edge in diagram

Figure 156 shows an association as a graphic edge in a Class diagram. It represents three UML elements and their relationships with one another and with the two classes: the association (UML type **Association**) **A_item_order item**), the two association ends (UML type **Property**) **order** and **itemline**, the relationships between the associations and their two association ends (UML property **Association::memberEnd**), and the relationships between the association ends and the two classes (UML property **TypedElement::type**).

The UML edges outlined thus represent totally different content.

The edges representing the package hierarchy in Figure 155 represent a hierarchy relationship between two groups. This relationship is not mapped using connection definitions in ARIS, but represents a direct reference from the subordinate group to the superior group.

The relationship between the comment and the class in Figure 156 represents the UML property **Comment::annotatedElement** and is thus a single connection definition.

The generalization in Figure 156 is mapped to a group that is subordinate to the derived class. The link to the base class is created using a connection definition. Thus the generalization edge represents a hierarchy relationship, a group, and a connection definition.

The association in Figure 156 represents a total of three groups and five connection definitions.

Classic connection occurrences in ARIS always represent a single connection definition. Therefore, they are not suitable for representing all edge types in UML diagrams. To cope with this, the ARIS object model has been extended with a new type of connection occurrence, which is only used by ARIS UML Designer and is capable of representing any content. The new type of edge presentations is also used in UML diagrams for edge presentations that actually represent a single connection definition.

4.3 The most important mappings from UML to ARIS

The table below shows the most important mappings from UML to ARIS.

UML	Represented in metamodel by meta element of type	Mapped to ARIS type	Condition
UML elements	Class	Group	The element's metaclass is not OpaqueExpression and is not a specialization of LiteralSpecification.
		Attribute	The element's metaclass is OpaqueExpression or a specialization of LiteralSpecification.
UML properties	Property	(Group hierarchy relationship)	For the meta property: isDerived=false isComposite=true type is a metaclass
		Connection definition	For the meta property: isDerived=false isComposite=false type is a metaclass
		Attribute	For the meta property: isDerived=false type is a data type
		(calculated at runtime)	For the meta property: isDerived=true
UML diagrams	Diagram*	Model	
Node presentations	NodeSymbol*	Object occurrence	
Edge presentations	EdgeSymbol*	New type of connection occurrence	
Lane presentations	LaneSymbol*	Object occurrence	

* In the official UML metamodel, no constructs exist for formal description of the graphical representation. These meta elements represent an ARIS-specific extension.

5 Linking business process and UML modeling

If you want to develop an IT system that provides optimum support for your company's business processes, it is useful to start with an analysis of the business processes, in order to derive the corresponding requirements for the IT system. ARIS provides optimum tools for doing this, as it provides integrated business process and UML modeling in a single repository and enables you to link business process and UML content with each other.

You have two fundamental ways of linking business processes and UML:

1. Assignment of UML diagrams to business process objects
2. Reusing business process objects in UML as UML elements

For both linking methods, as the user you have a free choice of which specific business process and UML types you want to link to one another. There are no rigid specifications for this in the ARIS Method.

You can navigate between the ARIS UML Designer and ARIS Architect/ARIS Designer applications with no problems. Double-clicking an ARIS model in the Explorer tree in ARIS UML Designer automatically launches ARIS Architect or ARIS Designer (if not already running) and opens the ARIS model. Conversely, an assigned UML diagram can be opened in ARIS UML Designer by double-clicking the corresponding assignment symbol in an ARIS model in ARIS Architect/ARIS Designer.

The two types of link and the various navigation options are described in more detail below.

5.1 Assignment of UML diagrams to business process objects

Business process objects are linked to UML diagrams using assignments of the **Navigation** type, as outlined in section 3.2.2.4Linked diagrams (elements).

You can create this assignment either in ARIS UML Designer or in ARIS Architect and ARIS Designer

5.1.1 Creating the assignment in ARIS UML Designer

A diagram is assigned to an ARIS object in ARIS UML Designer in the same way as assigning a diagram to a UML element.

First select the ARIS object in Explorer and then go to the **Linked diagrams** properties page. If you have not opened the **Explorer** tab, which displays the property pages on the right-hand side, but you are in the Designer component, first open the Properties dialog for the ARIS object. On the **Linked diagrams** properties page, click **Assign diagram**.

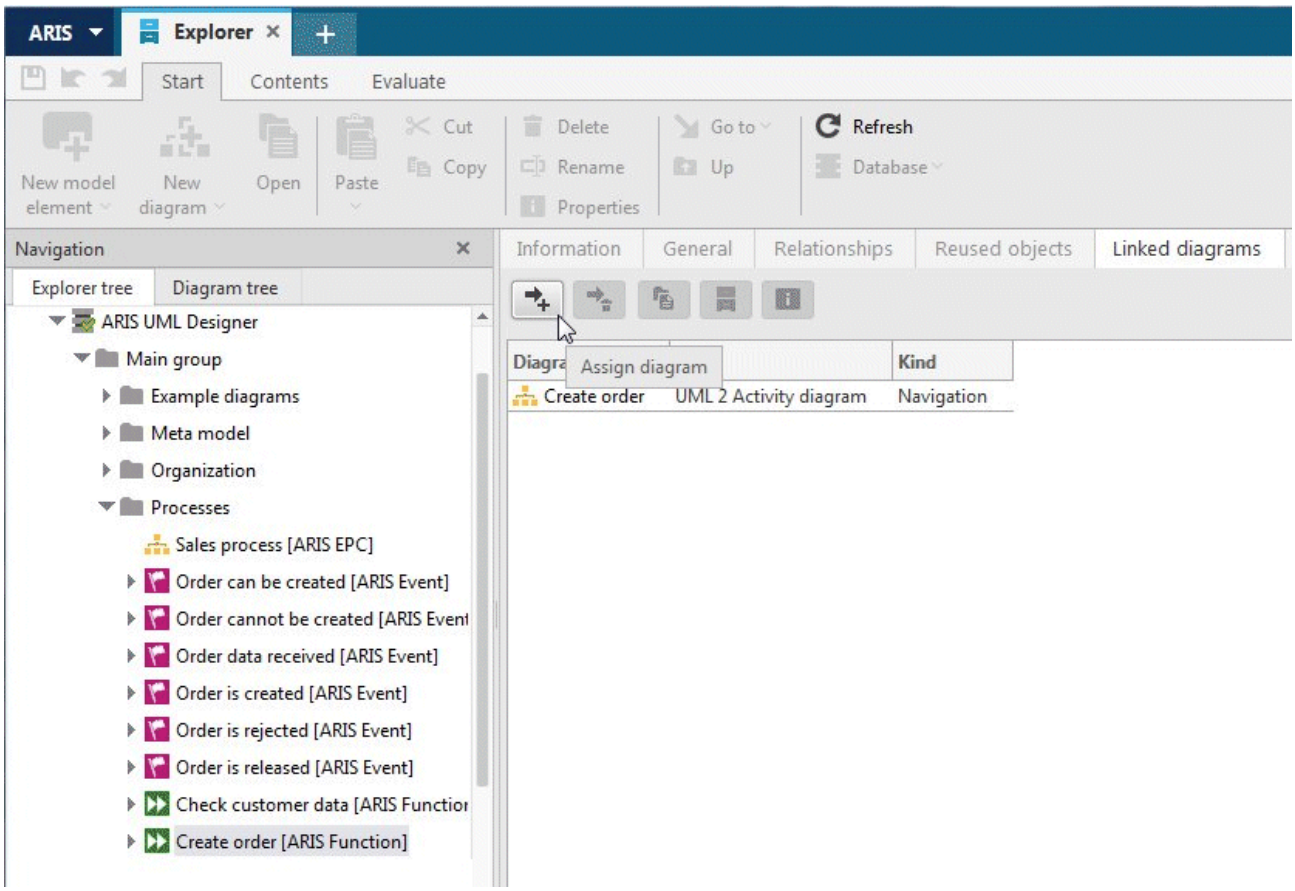


Figure 157: **Assign diagram** button

A dialog for selecting the diagram opens. You can either assign the diagram by searching the database or by selecting it directly in the Explorer tree (see Figure 158 and Figure 159).

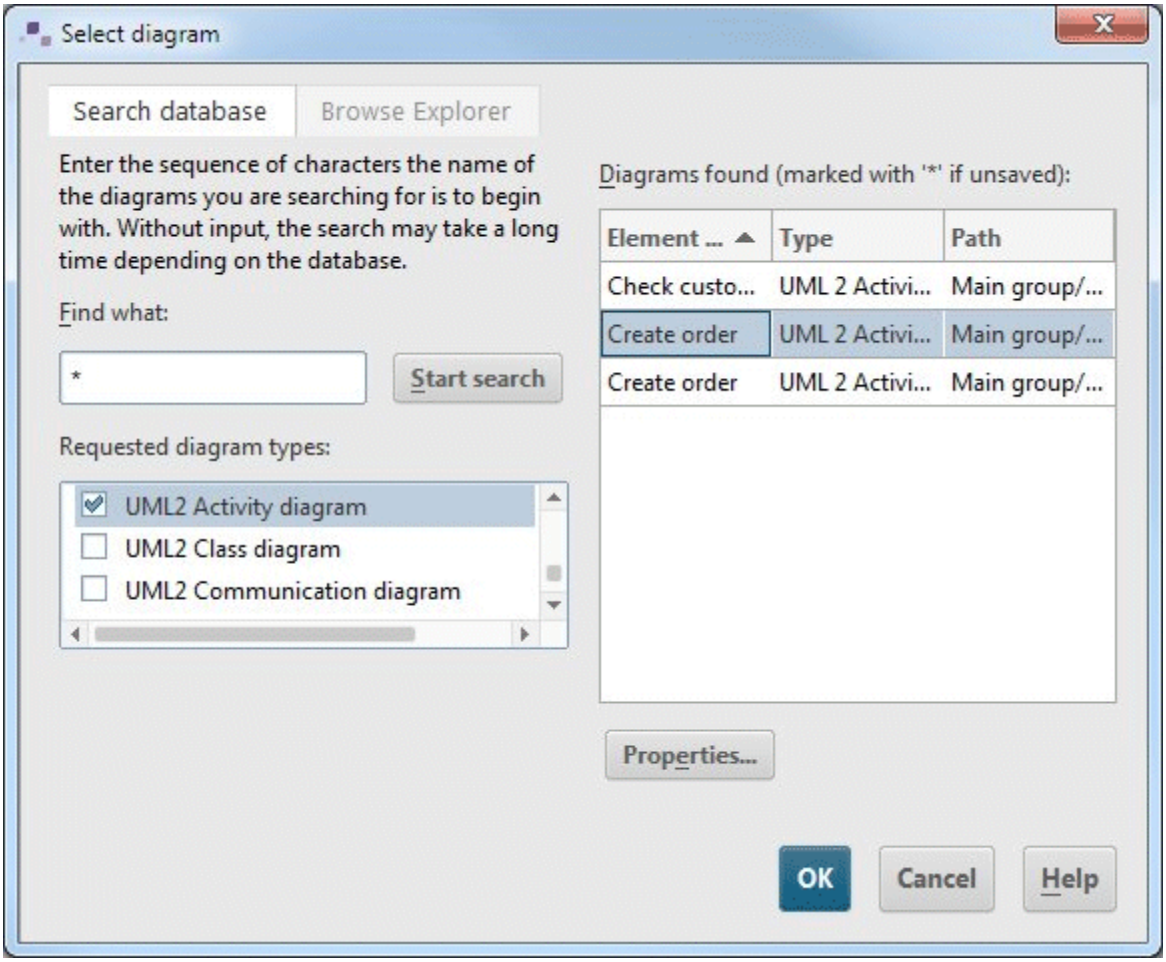


Figure 158: Diagram selection by searching in the database

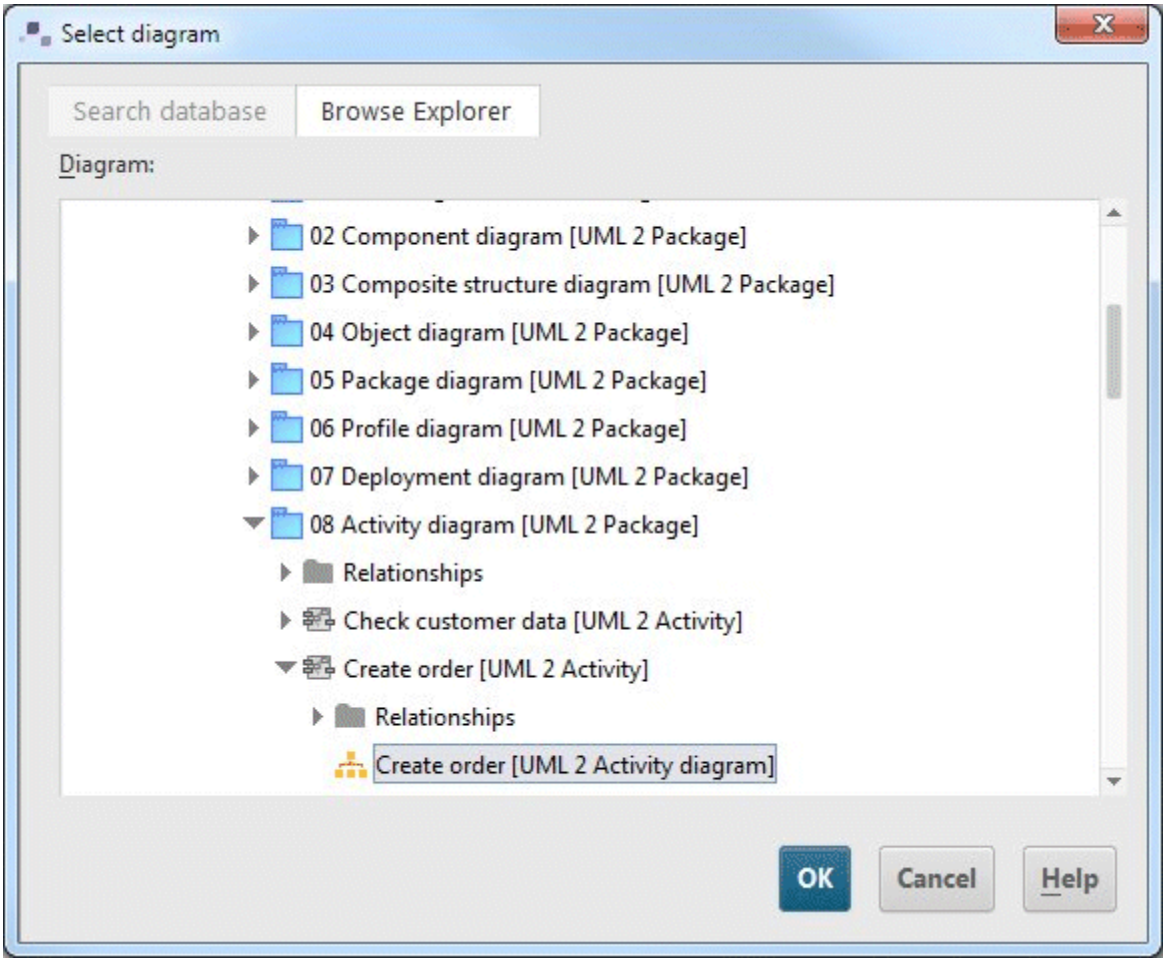


Figure 159: Diagram selection in Explorer

5.1.2 Creating the assignment in ARIS Architect and ARIS Designer

To assign a UML 2 diagram to an ARIS object in ARIS Architect or ARIS Designer, first select the object in the ARIS model and then click **Create assignment** in the **Start** tab bar.

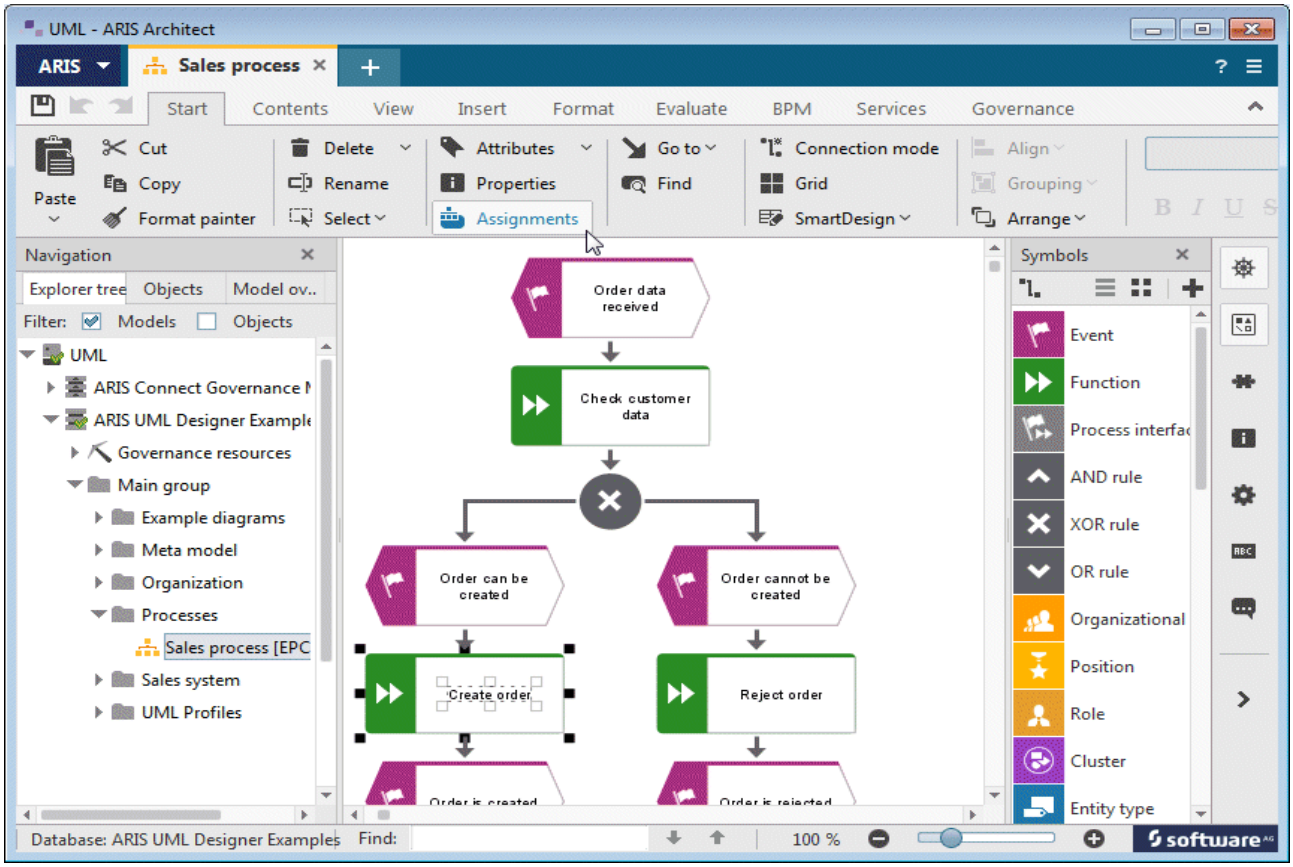


Figure 160: Launching the Properties dialog for a function in an ARIS model

The **Assignments** properties page opens. Click **New** to create a new assignment.

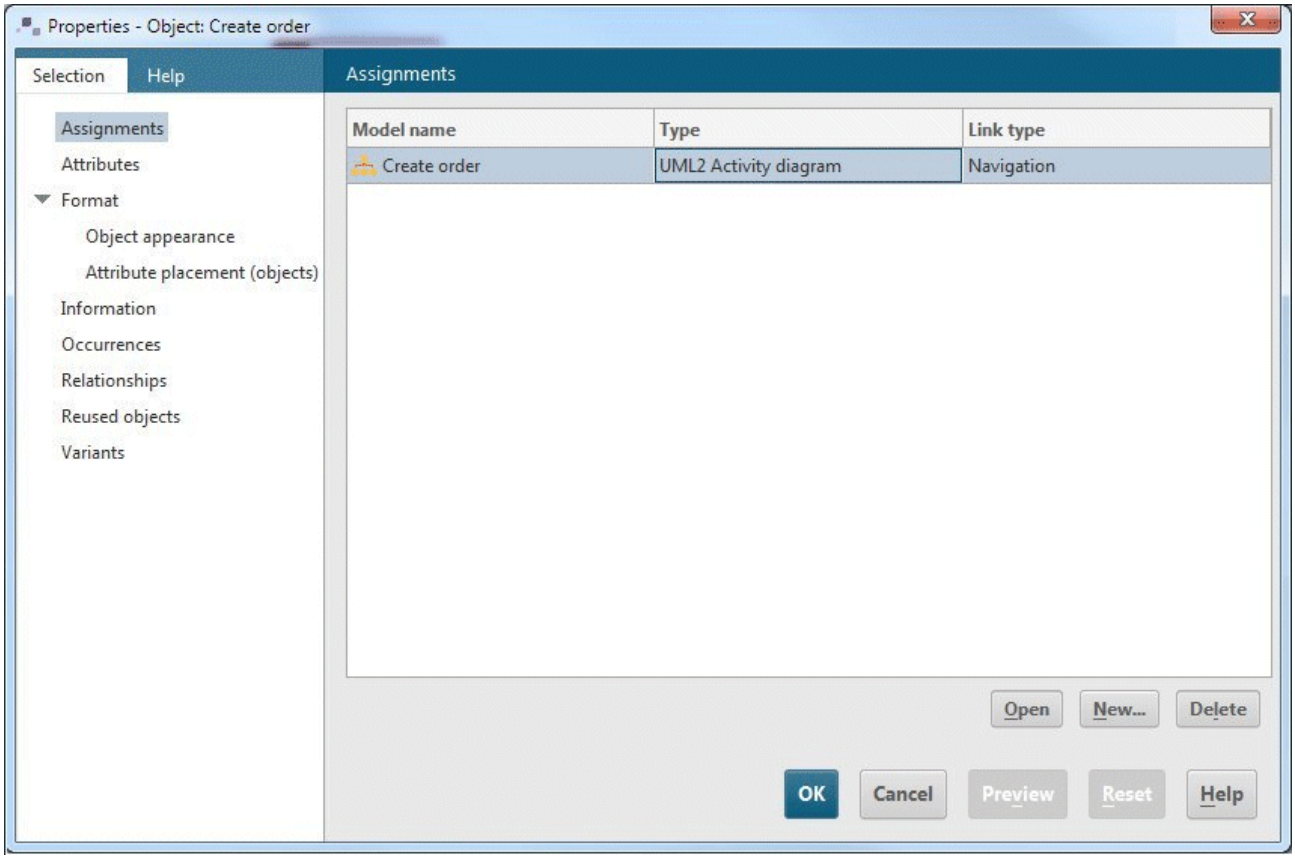


Figure 161: **Assignments** properties page in ARIS Architect and ARIS Designer

You are asked whether you want to assign an ARIS model or a UML 2 diagram. Click **UML -2 diagram**.

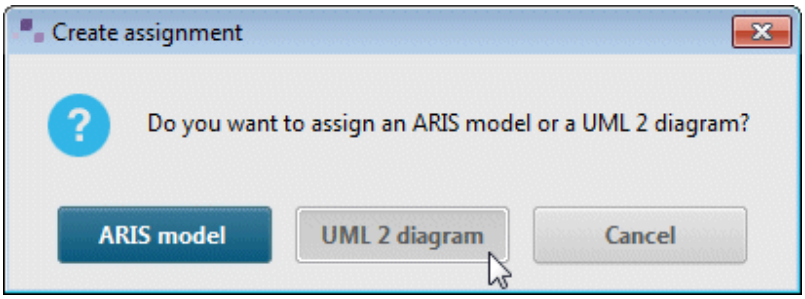


Figure 162: Query for diagram type to be assigned

In the subsequent dialog, you can select a UML diagram to assign.

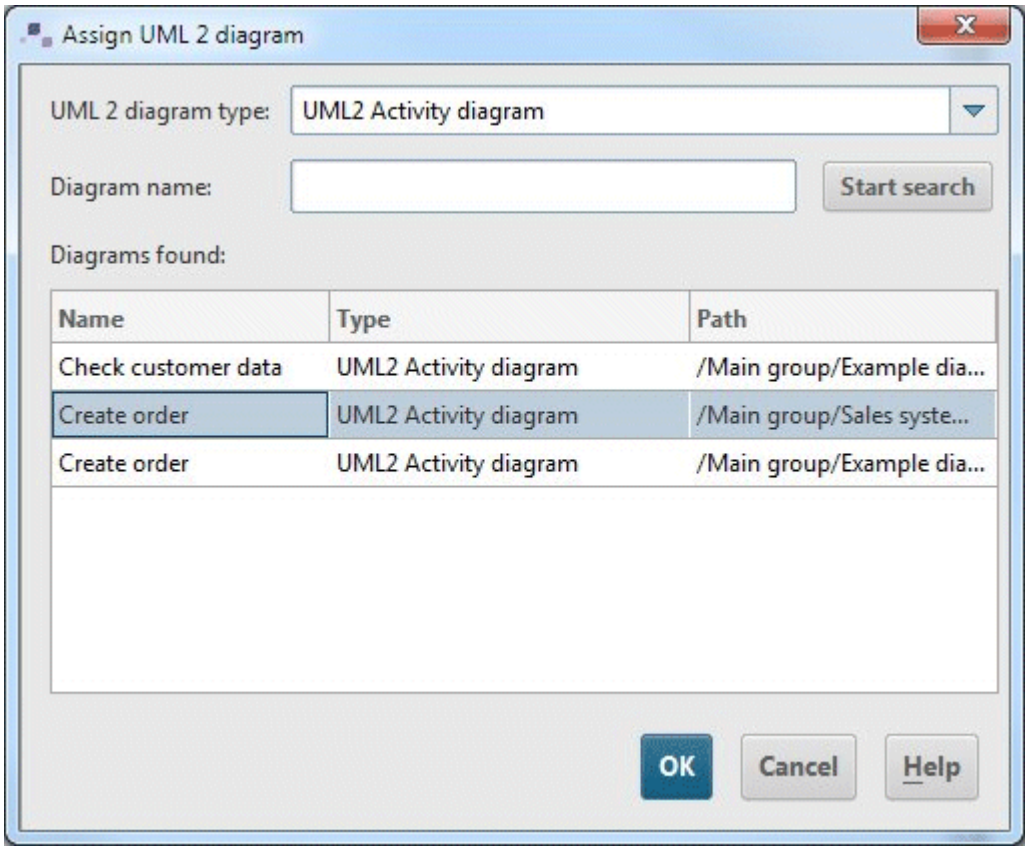


Figure 163: Dialog for selecting a UML diagram in ARIS Architect and ARIS Designer

The diagram selected here is added to the table of assigned diagrams (see Figure 164).

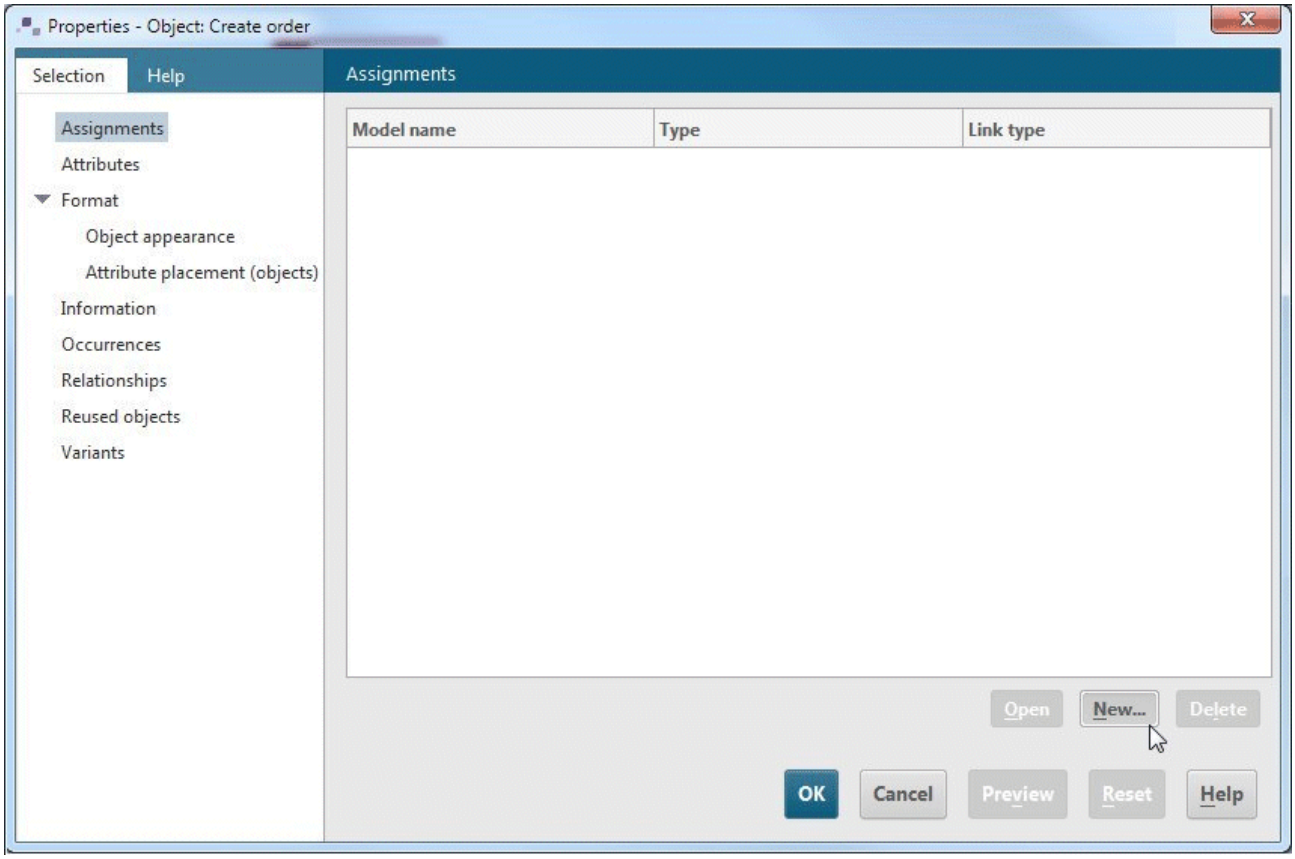


Figure 164: Assigned UML diagram in the Properties dialog for the ARIS object

In the ARIS model, an assignment symbol is displayed (see Figure 165).

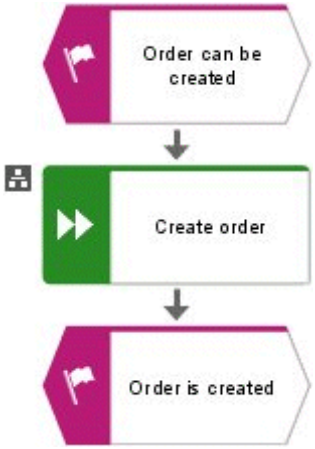


Figure 165Assignment symbol on the function

5.2 Reusing business process objects as UML elements

Before you can use business process objects in UML diagrams, you must specify which ARIS object types are to be mapped to which UML element types. You can then insert these ARIS objects in UML diagrams as UML elements.

The ARIS object has no direct presentation in the UML diagram. Instead, a new UML element is created and is linked to the ARIS object using a special reuse relationship.

The individual aspects of this reuse are described in more detail below.

5.2.1 Specifying the mapping of ARIS to UML types

The mapping of ARIS to UML types is defined in the **Link types** area of the **Administration** tab, which has been briefly outlined in section 0Link types. Note that this functionality is only available to you on the **Administration** tab in ARIS UML Designer, but not in ARIS Architect.

To define a mapping, select **Object links** under **Link types** in the Explorer tree on the **Administration** tab and then click **New > Object link definition** in the pop-up menu.

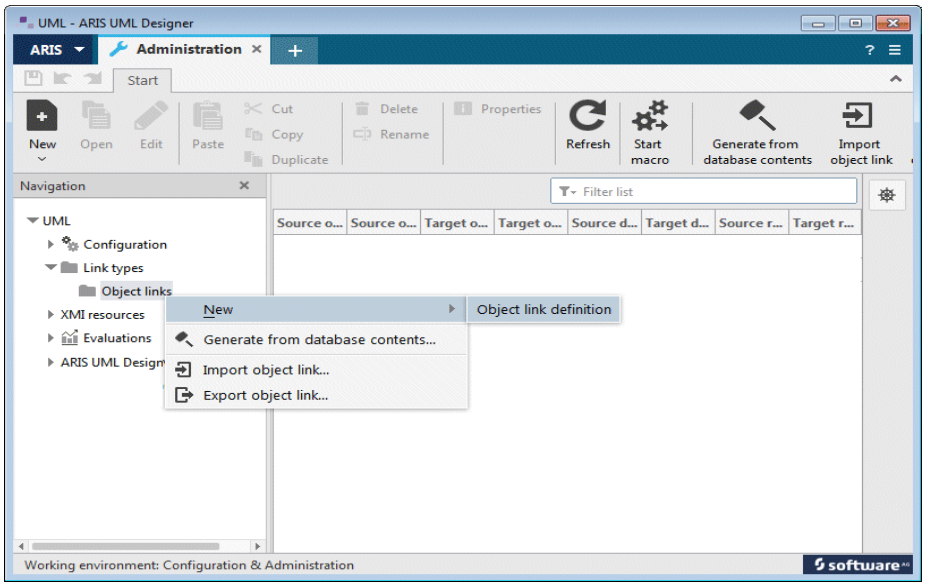


Figure 166: Creating a new object link definition

The **Create object link** dialog opens.

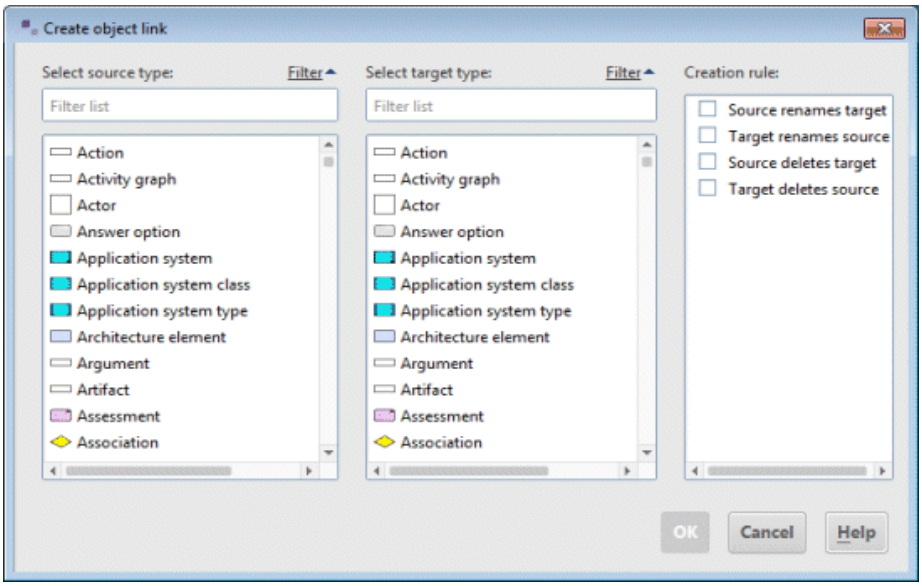


Figure 167: Dialog for creating a new object link definition

Select an ARIS object type in the **Select source type** column, and the relevant UML 2 element type in the **Select target type** column. It is theoretically possible to define mappings between any types, even between classic ARIS object types. However, ARIS Architect only supports mapping of classic ARIS object types to UML 2 element types, which means that you should always select a classic ARIS object type as the source type and a UML 2 element type as the target type.

You also have the option of enabling one or more of the following rules:

- **Source renames target**

  This rule means that when the ARIS object is renamed the UML element is also renamed so that it has the same name as the ARIS object.

- **Target renames source**

  This rule means that when the UML element is renamed the ARIS object is also renamed so that it has the same name as the UML element.

- **Source deletes target**

  This rule means that the UML element is deleted as soon as the underlying ARIS object is deleted.

- **Target deletes source**

  This rule means that the ARIS object is deleted as soon as the corresponding UML element is deleted.

Entering the initial letters of the type you are looking for in the input fields above the object lists filters the object list accordingly.

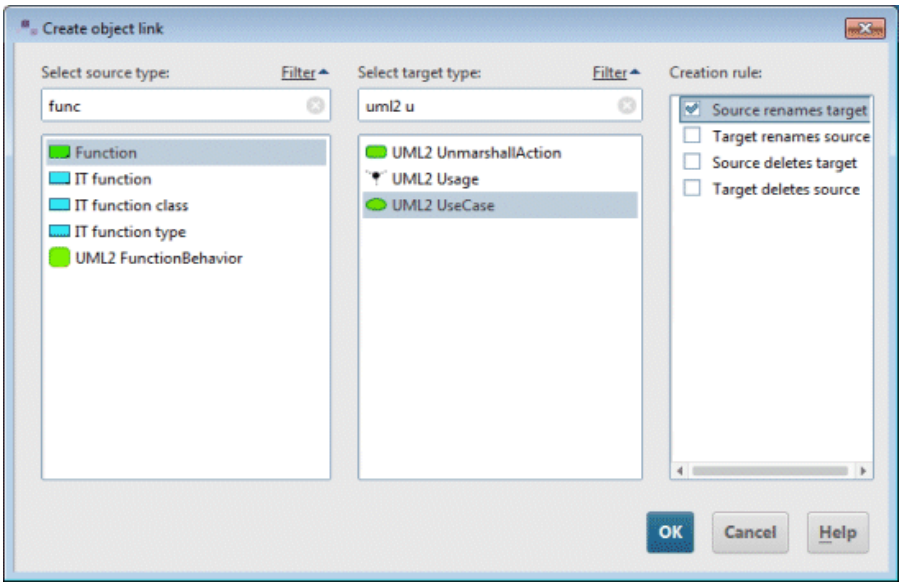


Figure 168: Definition of a mapping of the ARIS object type Function to the UML 2 element type UseCase

Figure 168 shows the definition of the mapping of the ARIS object type **Function** to the UML type **UseCase** with the rule that when the function is renamed the use case is given the same name. Clicking **OK** creates the definition and it is then displayed in the table.

Source object name	Source object type	Target object name	Target object type	Source deletes target	Target deletes source	Source renames target	Target renames source
Function	OT_FUNC	UML2 UseCase	OT_UML2_USE_CASE	✘	✘	✔	✘

Figure 169: A new object link definition created

This mapping enables you to view a function in a business process to be realized by an IT system as a use case for the purpose of object-oriented analysis and to reuse it as a UML element of the **UseCase** type.

You can use this method to map an ARIS object type to various UML element types, and multiple ARIS object types to a single UML element type.

For example, you could map the **Function** type not only to **UML2 UseCase** but also to **UML2 Operation**, enabling the function to also be reused in UML as an operation for a business class .

Conversely, it may be useful to map various ARIS types such as **Organizational unit**, **Person**, or **Application system** – i.e., all types that are linked to execution or monitoring of a function in some way – to the UML type **Actor**, so that you can reuse these ARIS objects as actors in UML use case diagrams.

Source object name	Source object type	Target object name	Target object type	Source deletes target	Target deletes source	Source renames target	Target renames source
Person	OT_PERS	UML2 Actor	OT_UML2_ACTOR	✘	✘	✔	✘
Function	OT_FUNC	UML2 Operation	OT_UML2_OPERATION	✘	✘	✔	✘
Entity type	OT_ENT_TYPE	UML2 Class	OT_UML2_CLASS	✘	✘	✔	✘
ERM attribute	OT_ERM_ATTR	UML2 Property	OT_UML2_PROPERTY	✘	✘	✔	✘
Application system type	OT_APPL_SYS_TYPE	UML2 Actor	OT_UML2_ACTOR	✘	✘	✔	✘
Technical term	OT_TECH_TRM	UML2 Class	OT_UML2_CLASS	✘	✘	✔	✘
Function	OT_FUNC	UML2 UseCase	OT_UML2_USE_CASE	✘	✘	✔	✘
Organizational unit	OT_ORG_UNIT	UML2 Actor	OT_UML2_ACTOR	✘	✘	✔	✘
Function	OT_FUNC	UML2 Action	OT_UML2_ACTION	✘	✘	✔	✘

Figure 170: Different object link definitions

Figure 170 shows the object link definitions described above.

When selecting the target type, you are not restricted to specific UML 2 types, i.e., those for which you can also create a directly corresponding UML element. The above example contains an object link definition in which the ARIS type **Function** is mapped to the abstract UML 2 type **Action**. In a case like this, when you insert a function in a UML Activity diagram, you are asked which specific action is to be created.

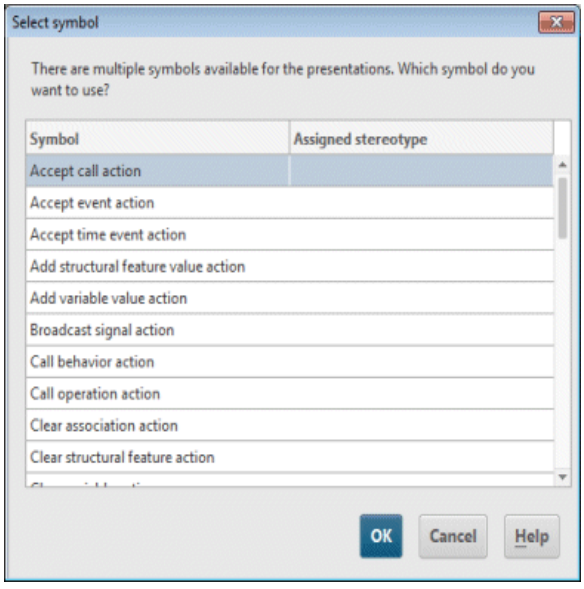


Figure 171: Symbol selection for abstract type

5.2.2 Reusing an ARIS object in a UML diagram

In the ARIS model, first select the ARIS object you want to reuse as a UML element. Copy it to the clipboard by clicking **Copy** in the pop-up menu.

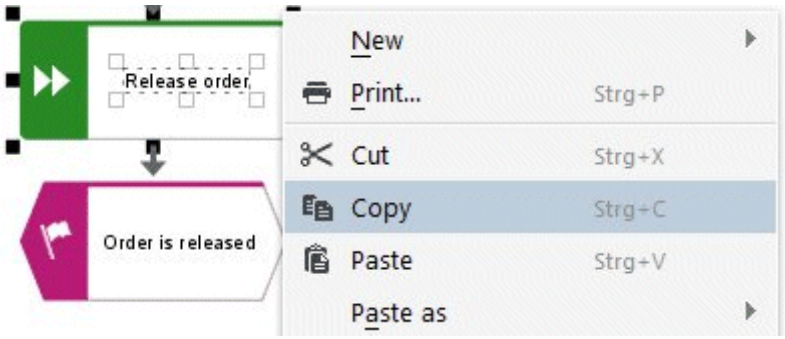


Figure 172: Copying the ARIS object to the clipboard

Alternatively, you can also select and copy the ARIS object in the Explorer tree in ARIS Architect, ARIS Designer, or ARIS UML Designer.

In ARIS UML Designer, open the pop-up menu by right-clicking the diagram background and then click either **Paste** or **Paste as > Place here as reused objects**.

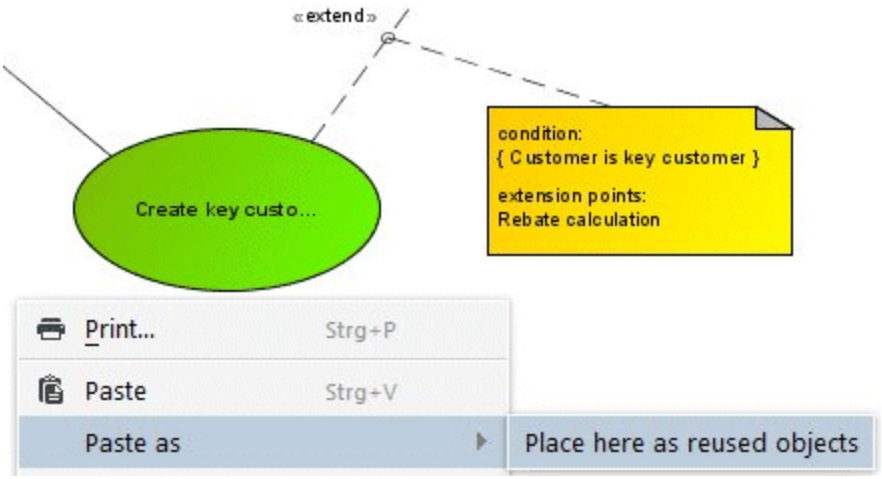


Figure 173: Pasting the ARIS object in the UML diagram

The **Reuse objects** dialog opens for selecting the underlying object link definition.

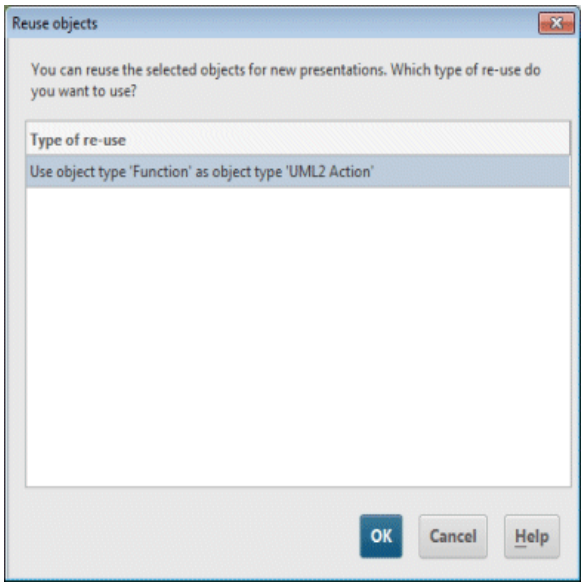


Figure 174: Selecting the object link definition

If there are several symbols that can be used for presentations of elements of the UML type in the diagram, you are asked which of the symbols is to be used.

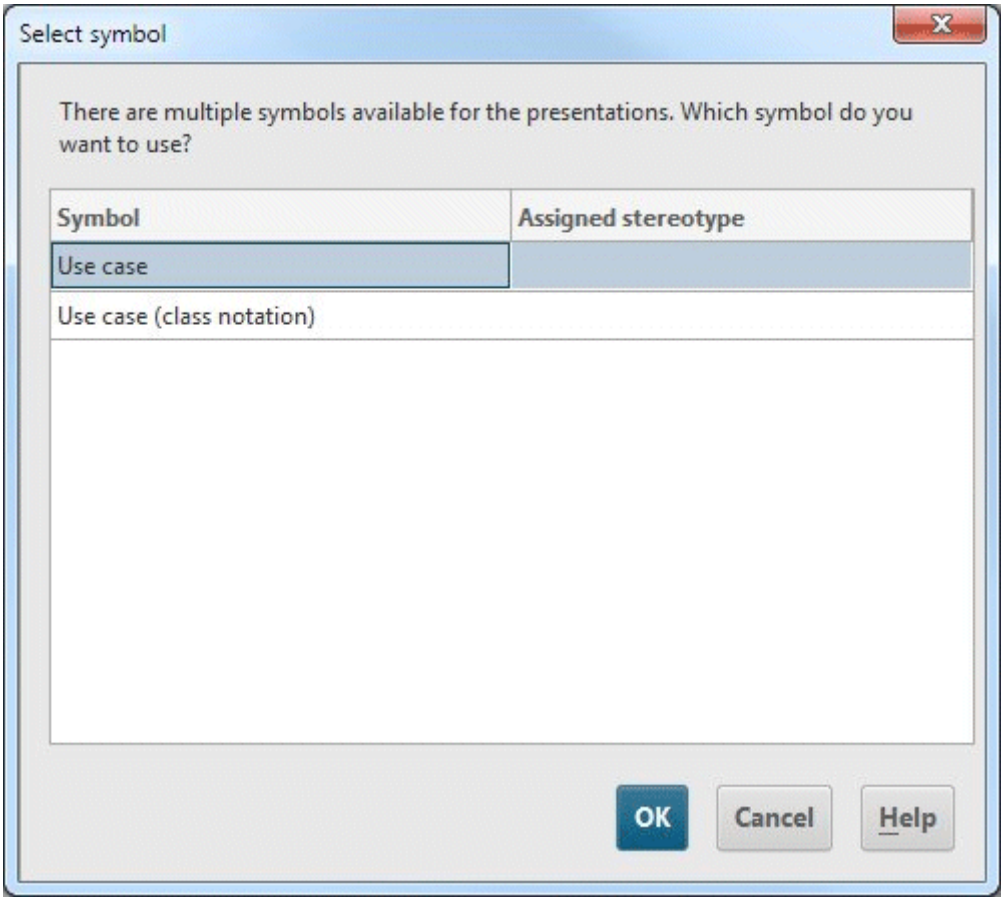


Figure 175: Dialog for selecting the symbol to be used

This creates a new use case in the diagram.

Figure 176: New use case created

The use case has the same name as the underlying function and displays it on its **Reused objects** properties page.

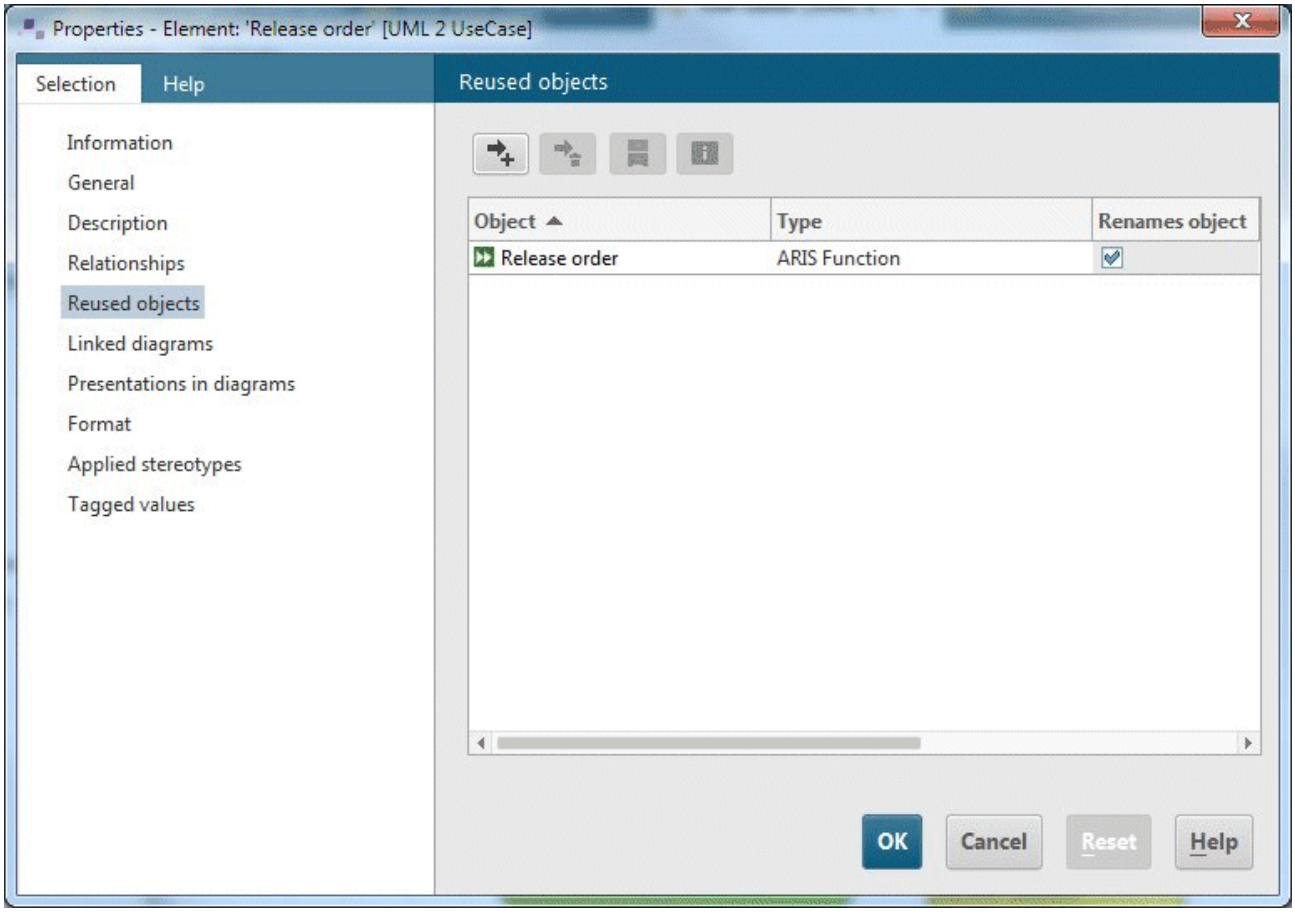


Figure 177: **Reused objects** properties page in ARIS UML Designer

After saving in ARIS UML Designer, this information is also available in ARIS Architect and ARIS Designer.

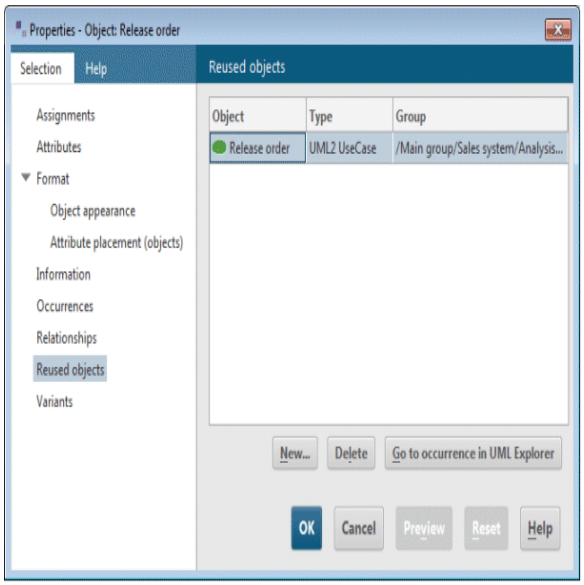


Figure 178: **Reused objects** properties page in ARIS Architect and ARIS Designer

5.2.3 Managing the object link definitions

The pop-up menu for an object link definition provides you with various options for editing.

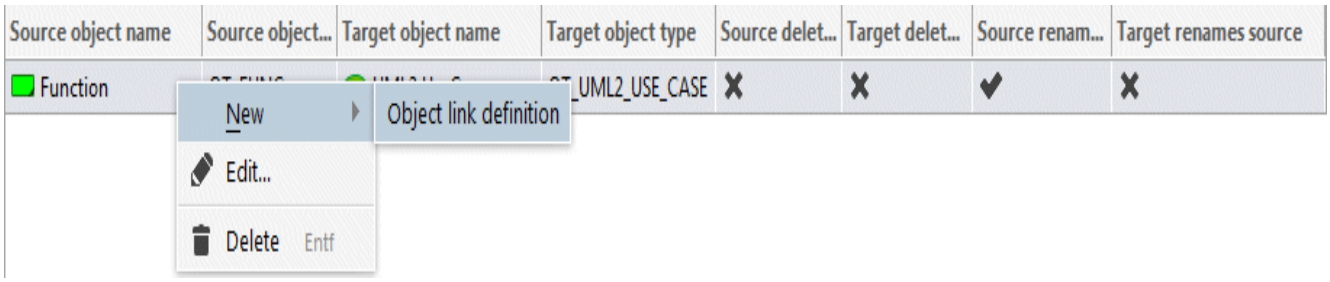


Figure 179: Pop-up menu items for editing object link definitions

You can create a new object link definition, edit the selected definition, or delete the selected definition. Clicking **Edit** opens the **Create object link** dialog, as when creating a new object link definition (see Figure 167).

The options for editing the selected object link definition can also be found in the tab bar.

Figure 180: Buttons in tab bar for managing object link definitions

The object link definitions are saved in the system database on the ARIS server. Selecting **Object links** in the **Link types** area in the Explorer tree on the **Administration** tab, you can export the object link definitions to a file to transfer them to a different ARIS server by clicking **Export object link**. To import, click **Import object link** on the other server.

Clicking **Generate from database contents** starts the analysis for all existing reuse relationships within an ARIS database and creates object link definitions for them in the system database.

This is helpful if you are importing a database with reuse relationships and you then want to create these reuse relationships yourself but you have not yet created the corresponding object link definitions. Merely to view or evaluate the reuse relationships in an ARIS database, it is not necessary to create the object link definitions in the system database.

5.3 Navigation between ARIS Architect / ARIS Designer and ARIS UML Designer

Although ARIS Architect or ARIS Designer and ARIS UML Designer are different programs, they provide easy options for navigating to elements and diagrams in the respective other program.

5.3.1 Navigation from ARIS Architect / ARIS Designer to ARIS UML Designer

In ARIS Architect and ARIS Designer, you can click the pop-up menu item **Go to > Occurrence in UML Explorer** for all groups, ARIS objects, ARIS models, and UML elements contained in the Explorer tree. Note that ARIS Architect and ARIS Designer only display the UML packages, models, and profiles that are located directly in an ARIS standard group, but not their content.

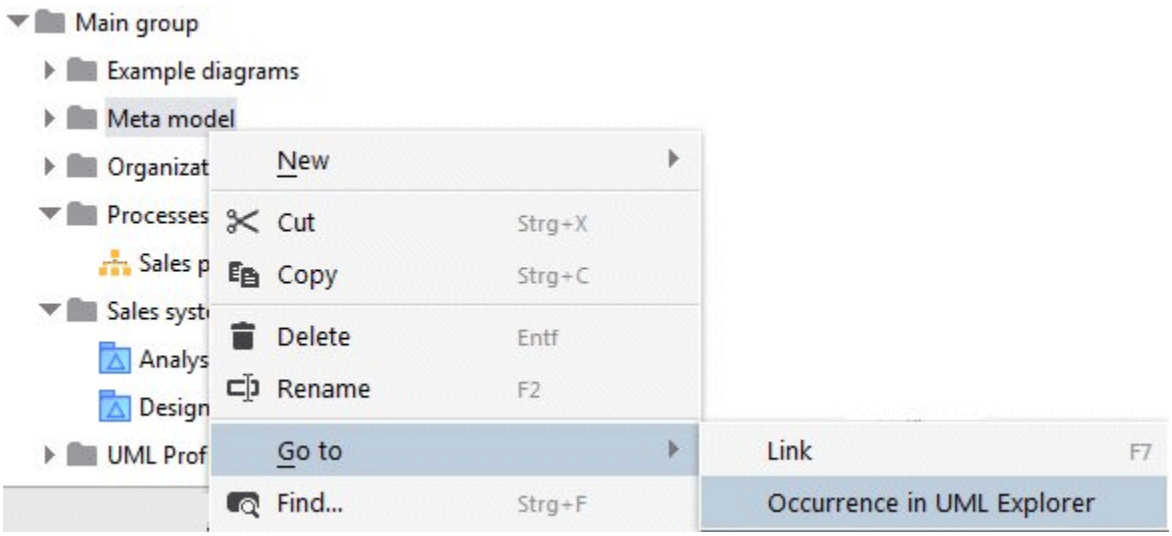


Figure 181: Pop-up menu item in ARIS Architect for navigation to ARIS UML Designer

After clicking the pop-up menu item, a check is made as to whether an ARIS UML Designer instance with the same server connection is running. If not, it is started. This is followed by a login to the same database in ARIS UML Designer, and the item selected in ARIS Architect or ARIS Designer is selected in the Explorer in ARIS UML Designer.

On the **Reused objects** properties page, it is also possible to click **Occurrence in UML Explorer** to navigate to the corresponding UML element in the Explorer in ARIS UML Designer (see Figure 178). This navigation option is also available in the pop-up menu.

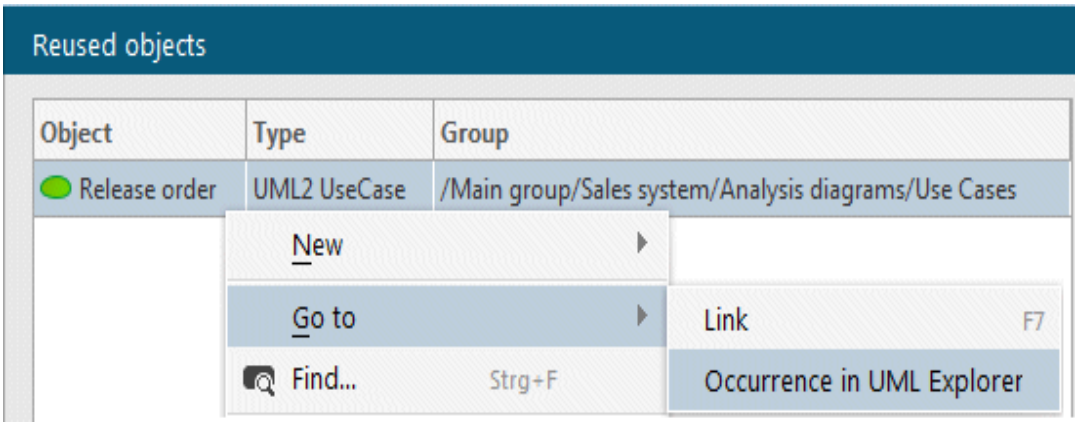


Figure 182: Pop-up menu item for navigation to the linked UML element

You can open assigned UML diagrams in ARIS Architect and ARIS Designer by double-clicking the assignment symbol in the ARIS model. You can also use the **Assignments** properties page and click the **Open** entry in the pop-up menu, or the **Open** button (see Figure 164), to open UML diagrams in ARIS UML Designer.

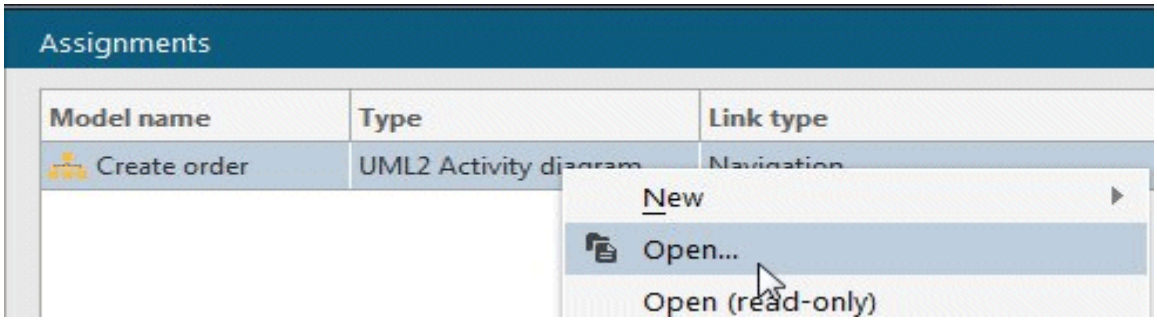


Figure 183: Pop-up menu item in ARIS Architect / ARIS Designer for opening a UML diagram

5.3.2 Navigation from ARIS UML Designer to ARIS Architect / ARIS Designer

In addition to the group hierarchy and the UML content, ARIS UML Designer displays all ARIS models and ARIS objects with their properties in the Explorer. To display and edit the ARIS models, it is necessary to switch to ARIS Architect or ARIS Designer.

For this purpose, for all groups, ARIS models, ARIS objects, and UML packages, models, and profiles that are directly located in a group, ARIS UML Designer provides you with a pop-up menu item for navigation to ARIS Architect or ARIS Designer.

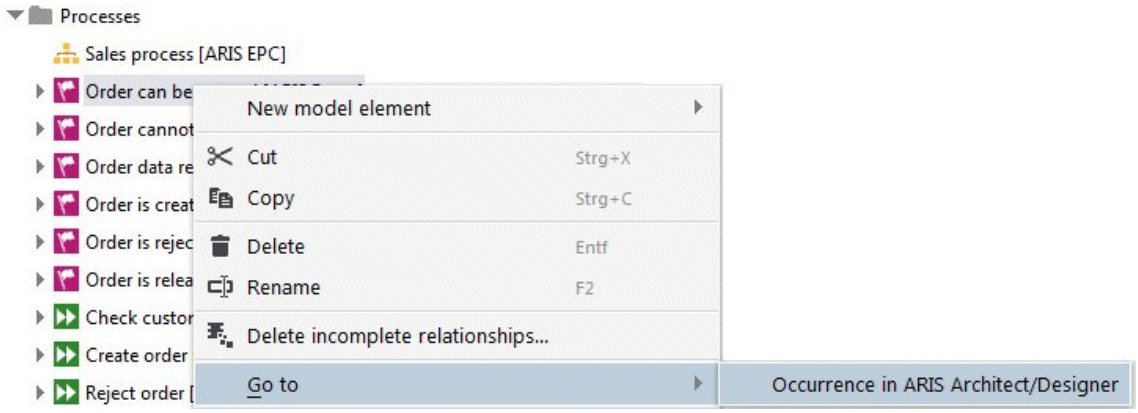


Figure 184: Pop-up menu item for navigation to ARIS Architect or ARIS Designer

The same functionality is available for a selected element in the **Start** tab bar.

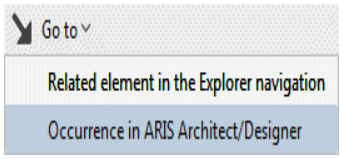


Figure 185: Tab bar item for navigation to ARIS Architect or ARIS Designer

After clicking the menu item, a check is made as to whether an ARIS Architect/ARIS Designer instance with the same server connection is running. If not, it is started. This is followed by a login to the same database in ARIS Architect or ARIS Designer, and the item selected in ARIS UML Designer is selected in the Explorer in ARIS Architect/ARIS Designer.

Just as for UML diagrams, the functionality for opening the diagram is provided in ARIS UML Designer for ARIS models.

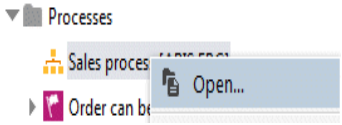


Figure 186: Opening an ARIS model in ARIS UML Designer

If required, an ARIS Architect/ARIS Designer instance is started and the ARIS model is opened in it.

The navigation is available not only in the Explorer but in all UML Designer components that display elements that are also visible in the Explorer in ARIS Architect and ARIS Designer.

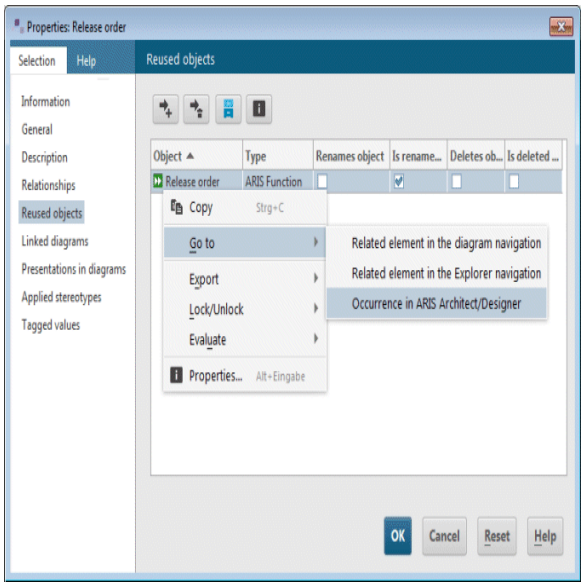


Figure 187: Navigation from UML element to underlying business process object

You also open assigned ARIS models using the **Linked diagrams** properties page for the UML element or from a diagram by double-clicking the assignment symbol on the presentation of the UML element.

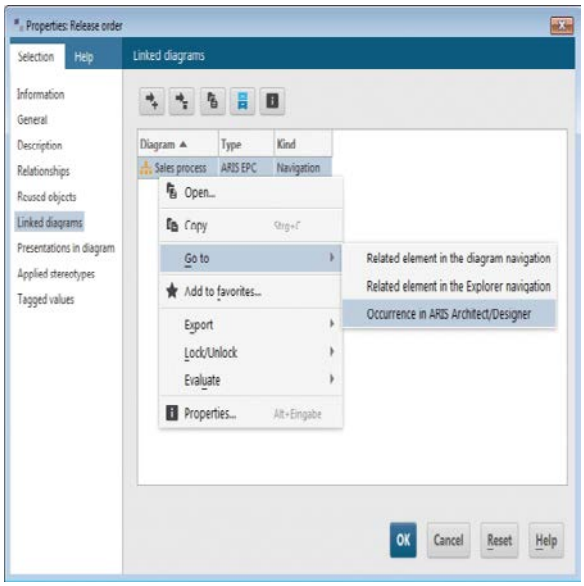


Figure 188: Menu items for navigation and opening an ARIS model assigned to a UML element

Figure 189: Assignment symbol for an ARIS model on a UML element

6 UML profiles

You can use UML profiles in ARIS UML Designer to extend the UML metamodel with user-defined types and properties and, at the same time, to reduce the number of UML types available in diagrams. Thus, UML profiles have a similar effect for UML modeling as ARIS method filters for ARIS standard modeling.

While an ARIS method filter is individually selected by the user when logging in to a database and applies to the entire database and the period of the login, UML profiles have fixed assignments to individual UML packages in the database. Therefore, they only apply to the packages (models and profiles) to which they are assigned, and to their content. This enables you to use different UML profiles in different areas of the database. In addition, UML profiles apply to all users in the same way, regardless of the method filter they have selected at login.

6.1 Predefined profiles in ARIS UML Designer

ARIS UML Designer contains several predefined UML profiles:

- **StandardProfile**

  This is the default profile from the UML specification. It contains all stereotypes defined in the UML standard.

- **OMF Meta Profile**

  This profile is used by the UML metamodel generator (see section 6.3The UML metamodel generator) to map the meta elements to UML.

- **OMF Extension Profile**

  This profile extends the profile modeling in ARIS UML Designer with options that are not available in the UML standard, e.g., with ARIS-specific properties such as multilingual text attributes and stereotypes for diagrams.

- **UML 1.4 Compatibility Profile**

  UML 1.4 is not completely forward compatible with UML 2. Some constructs and default stereotypes from UML 1.4 are no longer supported in UML 2. This profile is used in UML migration to ensure that these constructs are not lost during migration to UML 2.

6.2 Using profiles

6.2.1 Assignment of profiles to a package

Before you can use stereotypes of a profile, you must assign the profile to the package[29] within whose hierarchy the stereotypes are required.

The assignment is made on the **Applied profiles** properties page by clicking **Apply profiles**.

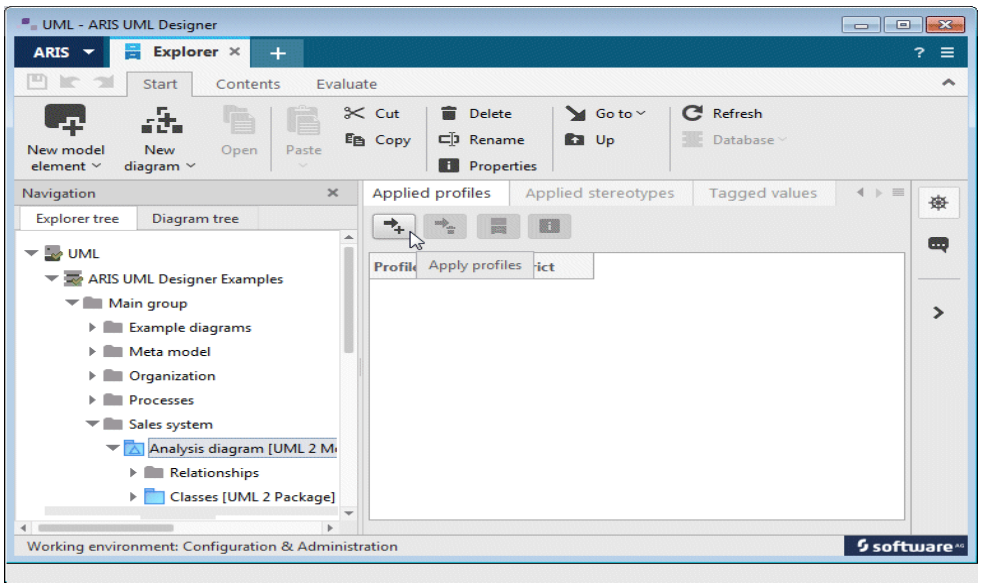


Figure 190: **Applied profiles** properties page

[29] The term **package** here also includes the two UML types **Model** and **Profile**, which are special types of packages.

The **Applied profiles** dialog is opened and contains the profiles to be assigned to the package.

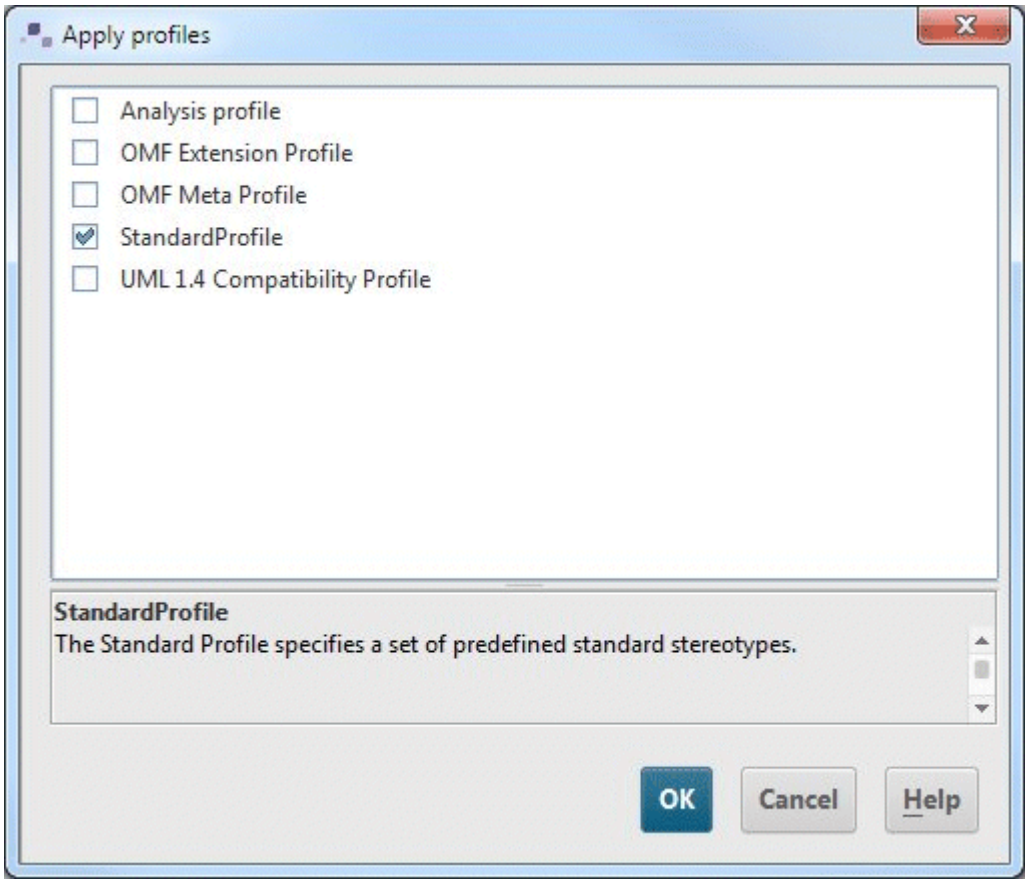


Figure 191: Profile selection dialog

The dialog lists all profiles that can be assigned. Enable the corresponding check box to indicate that the profile is to be assigned to the package.

If a profile is already assigned to the package, its check box is enabled automatically. In this case, you can disable the check box to indicate that the assignment of the profile to the package is to be removed.

When you select the profile in the dialog by clicking the text, a description of the profile is shown in the lower section of the dialog. The same description is also displayed as a tooltip if you hover the mouse pointer over a profile name for a short time.

Clicking **OK** assigns the selected profiles to the package.

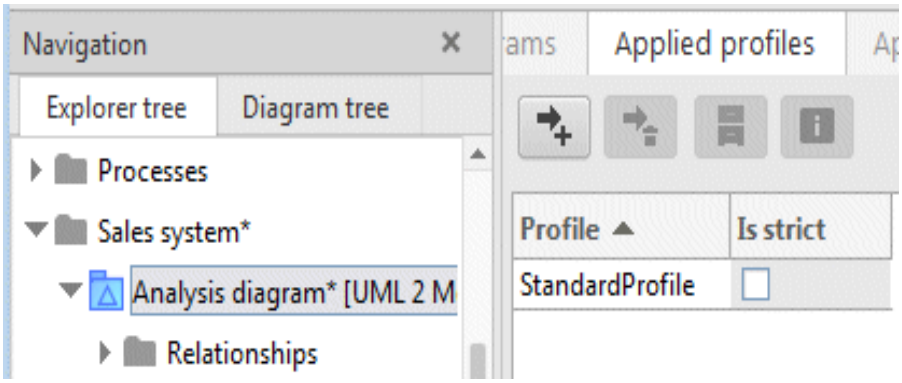


Figure 192: Package with assigned profile

By enabling the **Is strict** check box in Figure 192 you can specify that restrictions that define the profile cannot be canceled by other assigned profiles[30].

As soon as a profile has been assigned to a package, its stereotypes are available in the package hierarchy.

[30] The predefined profiles contain no restrictions.

6.2.2 Assignment of stereotypes to a UML element

Stereotypes are assigned to an element on its **Applied stereotypes** properties page by clicking **Apply stereotypes**.

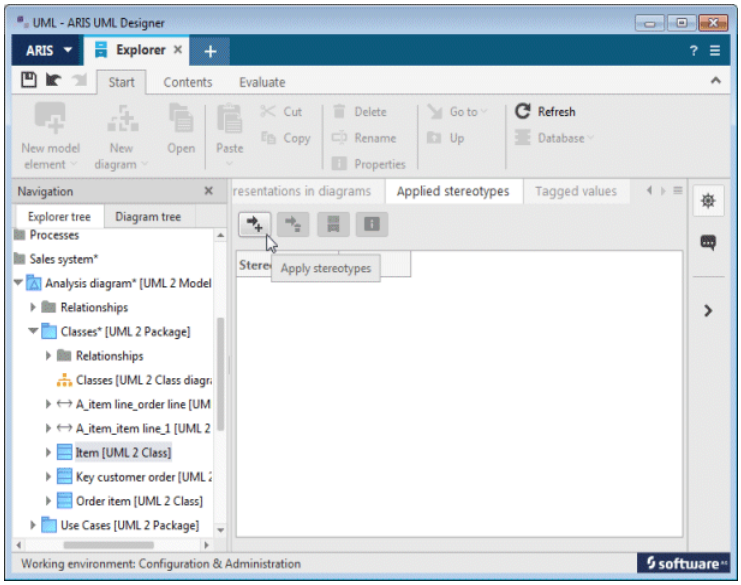


Figure 193: **Applied stereotypes** properties page

The **Apply profiles** dialog is opened and contains the stereotypes to be assigned to the element.

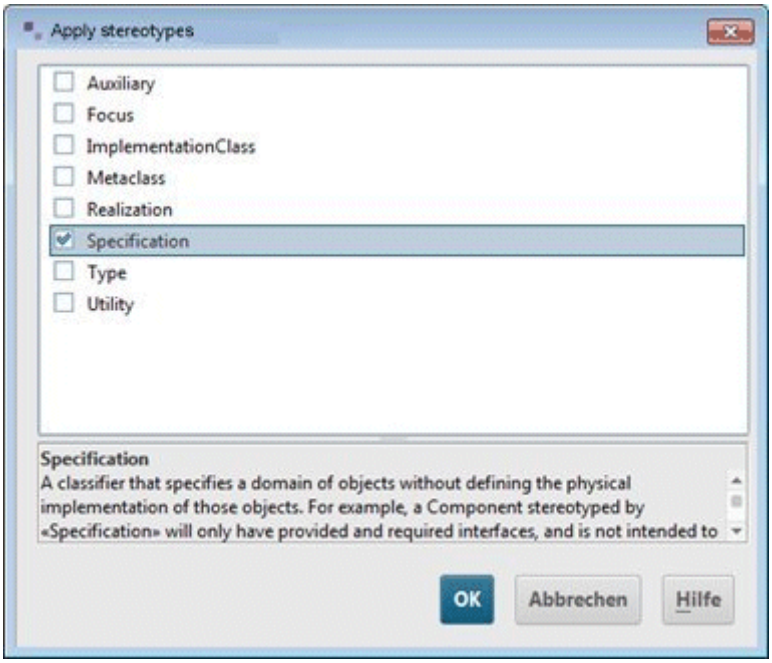


Figure 194: Dialog for stereotype selection

The dialog lists all stereotypes that can be assigned. Enable the corresponding check box to indicate that the stereotype is to be assigned to the element.

If a stereotype is already assigned to the element, its check box is enabled automatically. In this case, you can disable the check box to indicate that the assignment of the stereotype to the element is to be removed.

When you select the stereotype in the dialog by clicking the text, a description of the stereotype is shown in the lower section of the dialog. The same description is also displayed as a tooltip if you hover the mouse pointer over a stereotype name for a short time.

Clicking **OK** assigns the selected stereotypes to the element.

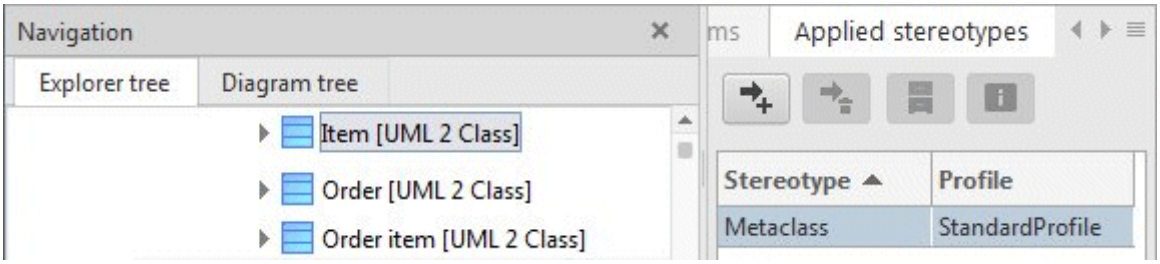


Figure 195: UML class with assigned stereotype

Stereotypes can also be assigned for multiple elements at the same time. To do this, select the corresponding elements in the Explorer tree or in the diagram and display their properties.

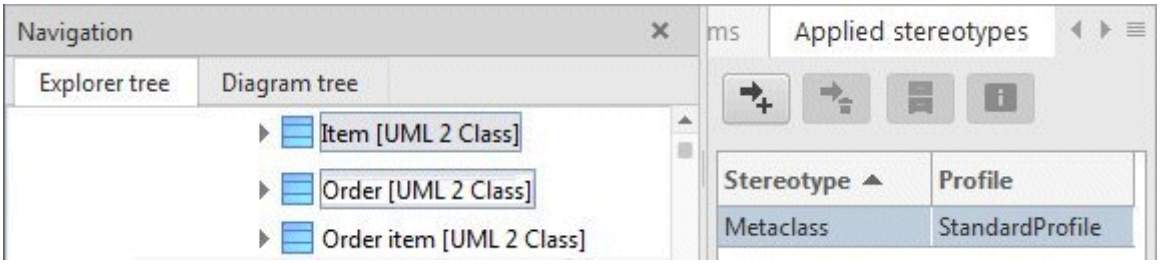


Figure 196: **Applied stereotypes** properties page for two selected classes

All stereotypes assigned to the selected elements are displayed. If a stereotype is not assigned to all selected elements, the entry is displayed in gray instead of black text in the table. You can click **Apply stereotypes** to simultaneously add stereotypes to all selected classes.

The stereotypes of the default profile do not define any new properties. The subsequent section on user-defined profiles explains how to display and edit the corresponding property values.

6.2.3 Stereotypes in the Symbols bar in diagrams

Assigning a profile to a package affects the **Symbols** bar in the diagrams contained in the package. For each symbol for whose metaclass the profile defines a stereotype, an additional symbol with the name of the stereotype is provided.

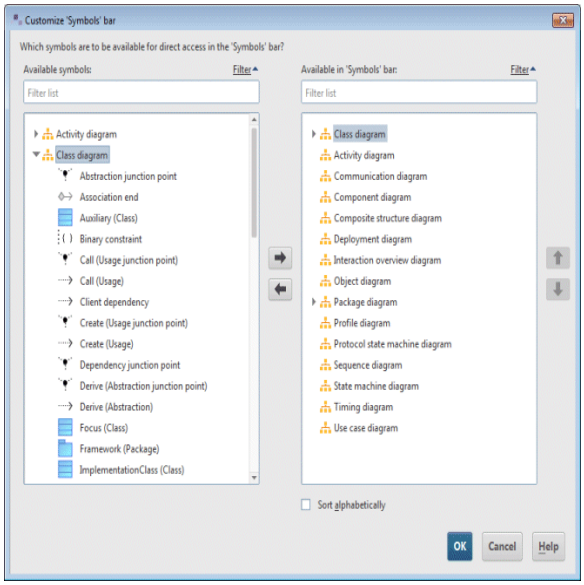


Figure 197: Configuration dialog for the **Symbols** bar with stereotype symbols

Figure 197 shows the configuration dialog for the **Symbols** bar for a Class diagram, whose package has been assigned the default profile. In addition to the usual symbols, it also provides symbols for the corresponding stereotypes. The underlying symbol name is displayed in brackets after each stereotype name.

For example, if you add the **Auxiliary (Class)** symbol to the **Symbols** bar for this kind of Class diagram and create an element in the diagram for this symbol, a new class is created in the diagram and this class is automatically assigned the **«Auxiliary»** stereotype.

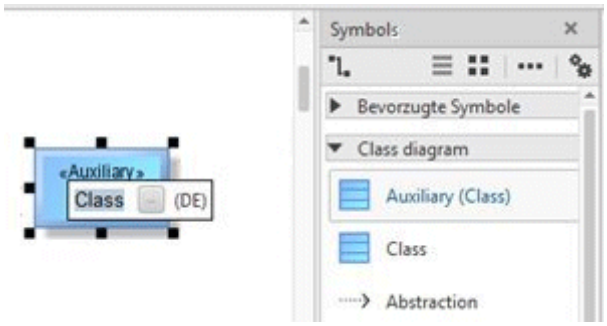


Figure 198: Creating a stereotyped class using a corresponding symbol

6.3 User-defined UML profiles

ARIS UML Designer supports creation of user-defined UML profiles using UML Profile diagrams The profiles are available immediately after creation in the database in which they are modeled. If you want to use this kind of user-defined profile in a different database, you can transfer it to other databases using standard ARIS functionalities such as Merge or XML export and import.

6.3.1 The UML metamodel generator

Profile modeling in ARIS UML Designer is carried out in line with the UML specification, i.e., both the metamodel and its metaclasses are represented graphically in the Profile diagram and related to the elements from the profile using edges.

In order to create a profile yourself, you first require the UML metamodel with its metaclasses. You can create the UML metamodel in the database using the metamodel generator.

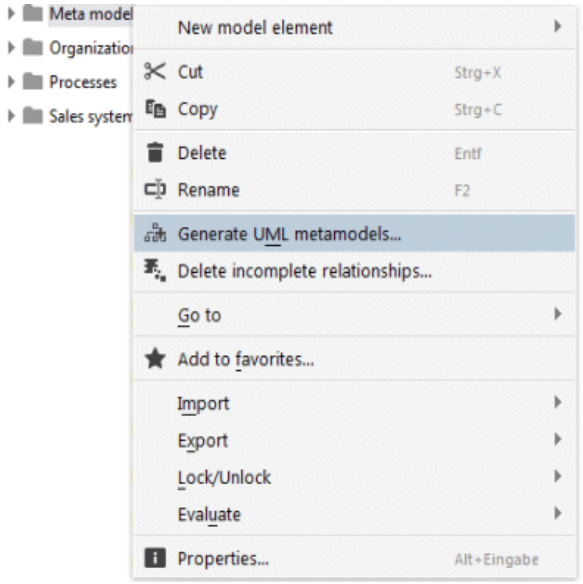


Figure 199: Launching the UML metamodel generator in the pop-up menu for a group

You can launch the metamodel generator in the pop-up menu for an ARIS group by clicking **Generate UML metamodels**. Alternatively, you can click the button of the same name in the **Content** tab bar.

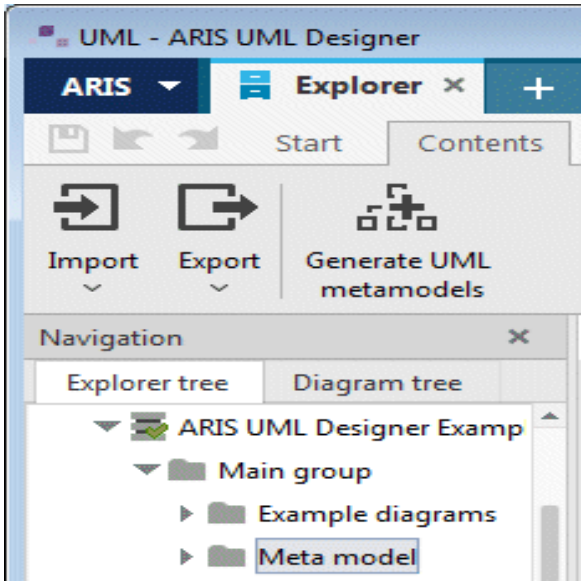


Figure 200: Launching the UML metamodel generator in the tab bar for a selected group

The dialog for generating the UML metamodel enables you to specify the language in which the metamodel is generated in the database.

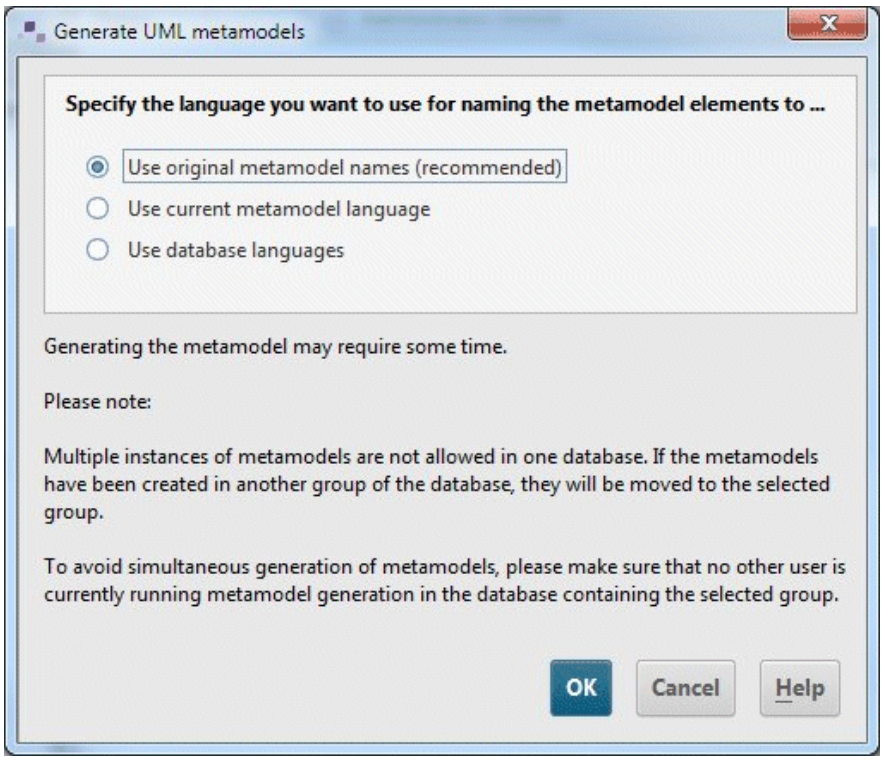


Figure 201: UML metamodel generator

The following options are available:

- **Use original metamodel names (recommended)**

  In this case, the original names from the UML metamodel are used. This is recommended because the UML specification only exists in English. While some metaclasses are relatively easy to identify if their names are in other languages, with other metaclasses it is much more difficult.
  With this option, the names are only created in the alternative language in the database, which means that you will always see the original metamodel names regardless of the database language you select at login.

- **Current metamodel language**

  This option creates the names in the language you have selected as the metamodel language in the general options (see section 3.4.2UML - General). Once again, the names are only created in the alternative language in the database. For example, if you have selected **German** as the metamodel language and **English** as the alternative language in the database, the German metamodel names are saved in the database language **English**.

- **Use database languages**

  Generation is carried out for each database language in the corresponding language, i.e., if you are logged in with the database language **German**, the metamodel names are displayed in German, and if you are logged in in English, the original names are displayed.

The metamodel generator creates three metamodels in the database.

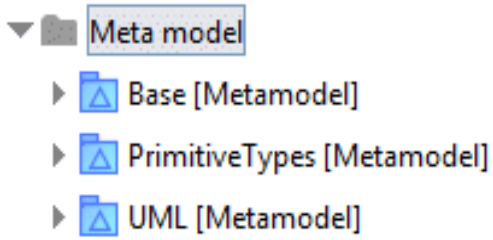


Figure 202: Generated metamodels

The **PrimitiveTypes** metamodel defines the primitive data types used by the UML metamodel. It is also part of the official UML standard from the OMG.

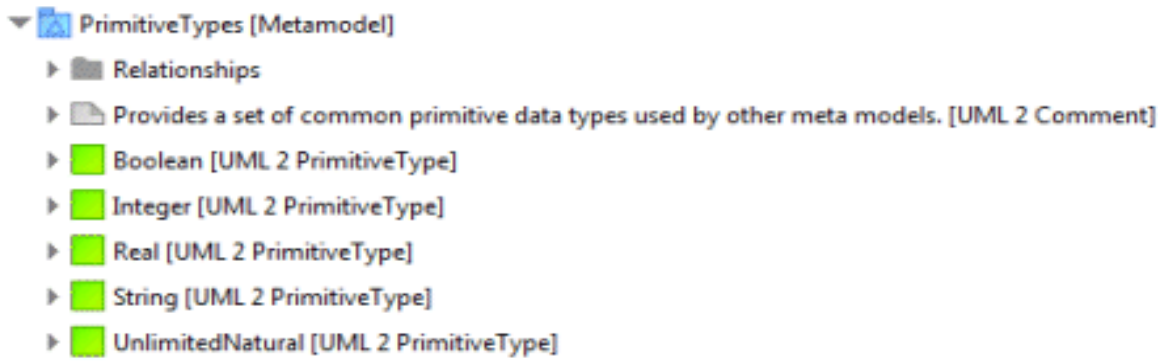


Figure 203: **PrimitiveTypes** metamodel

The **Base** metamodel is an ARIS-specific extension, which supplements the UML elements and diagrams with some ARIS-specific properties (see also section 2.4.1.1Root).

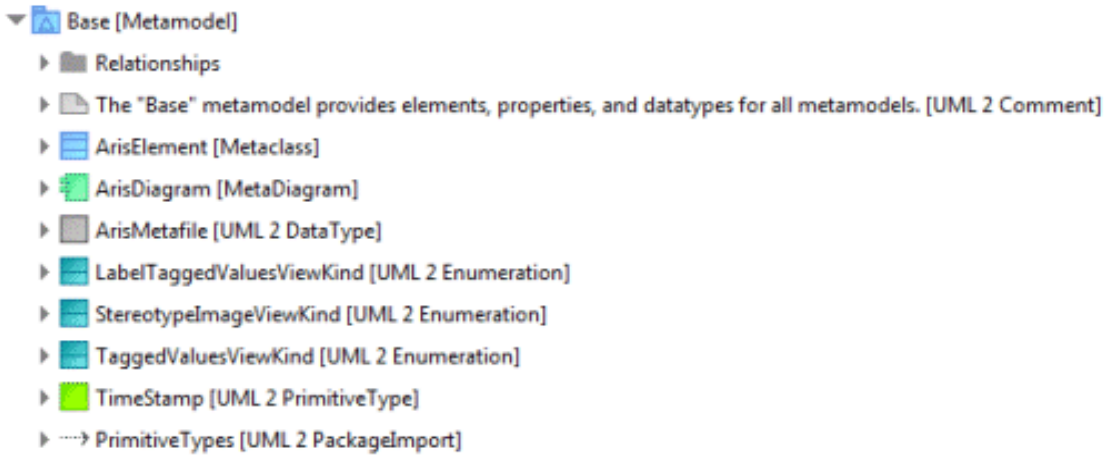


Figure 204: **Base** metamodel

The **UML** metamodel contains all UML metaclasses, associations, and properties and – as an ARIS-specific extension – the corresponding symbol and diagram types that are also relevant for modeling of profiles.

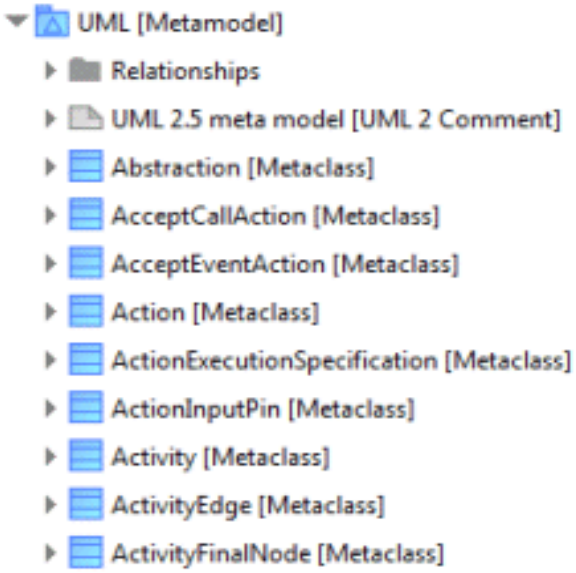


Figure 205: Small extract from the generated UML metamodel

Note that you can only use metamodels generated in this way for profile modeling. ARIS UML Designer identifies the relevant meta elements based on their GUID and not using their name, which can differ considerably depending on the generator settings.

The fact that each meta element is created with a specific GUID and a GUID in an ARIS database can only be used by one element results in certain consequences for generation.

Each generated meta element can only exist once in a database. Therefore, the metamodel generator always searches the database for an existing element with the corresponding GUID first, before it creates a new one. If it finds one, this element is used.

This means that if a generated metamodel already exists in a different group in the database, it is moved into the group in which the generator has been started.

If you move a meta element from the metamodel to another UML package, rename it, or edit it in some other way – which is explicitly **not recommended** – this meta element is restored to its original state the next time the metamodel generator is run.

6.3.2 Creating a profile

First create a new profile in an ARIS group in the Explorer tree and give it the name of your choice.

Figure 206: Creating a new profile in the Explorer tree

Then create a Profile diagram in the profile.

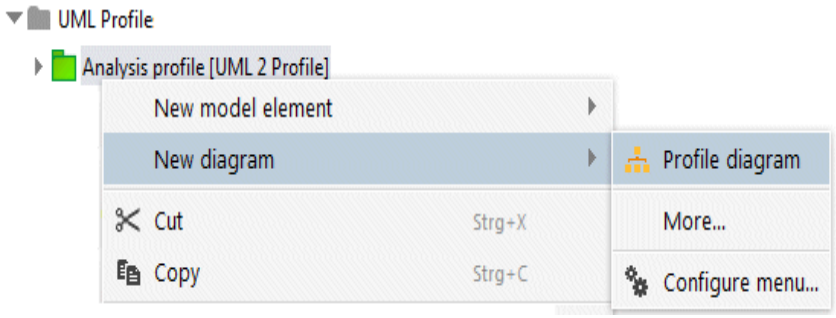


Figure 207: Creating a new Profile diagram

It is important that you first create the profile and then the Profile diagram in the profile, and not vice versa. If you create new elements in a diagram, they are created in the element in which the diagram is contained. The fact that the profile diagram is contained in the profile ensures that all stereotypes, Extension relationships, and enumerations that you create in the diagram as part of profile modeling are actually contained in the profile and not in another package.

Next drag the profile from the Explorer tree into the diagram.

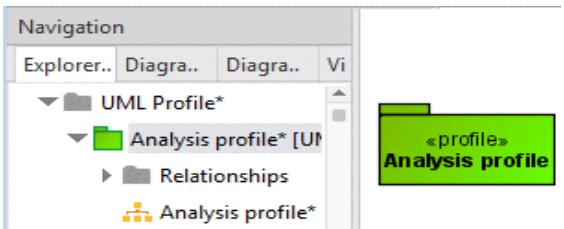


Figure 208: Presentation of the profile in the Profile diagram

Finally, you have to drag the UML metamodel created by the metamodel generator into the Profile diagram and create a relationship of the **Metamodel reference** type from the profile to the metamodel.

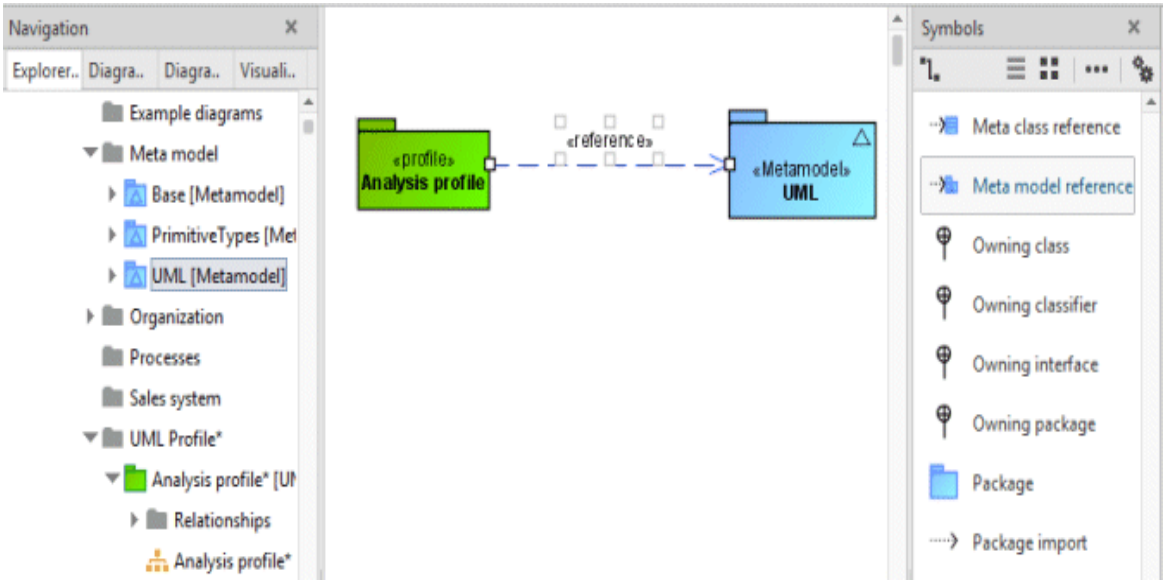


Figure 209: Definition of a pure extension profile

You have thus indicated that the profile fully supports **UML**. Section 6.3.7Creating a filter profile outlines how to define profiles that only support a subset of UML.

If you enter a description in the properties of the profile, it is displayed later when assigning the profile to a package in the profile selection dialog.

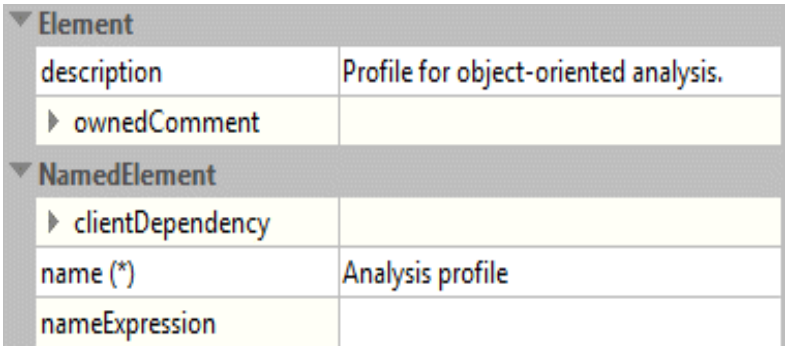


Figure 210: Description of the profile

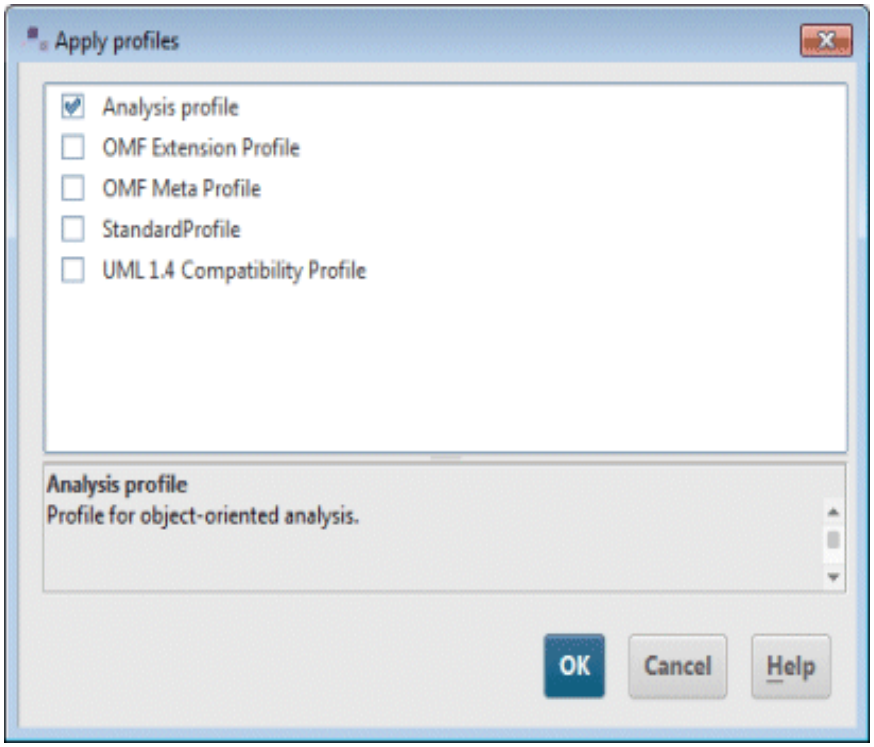


Figure 211: User-defined profile with description in the profile selection dialog

6.3.3 Creating a stereotype

Stereotypes are used to express particular semantics that are not provided in the UML standard, e.g., the **«Business class»** and **«Design class»** stereotypes for illustrating the significance of the corresponding classes, or to define new properties. The stereotypes of the UML default profile fall into the first category. They do not define any new properties.

If you want to define a new stereotype, you must first create it in the Profile diagram[31].

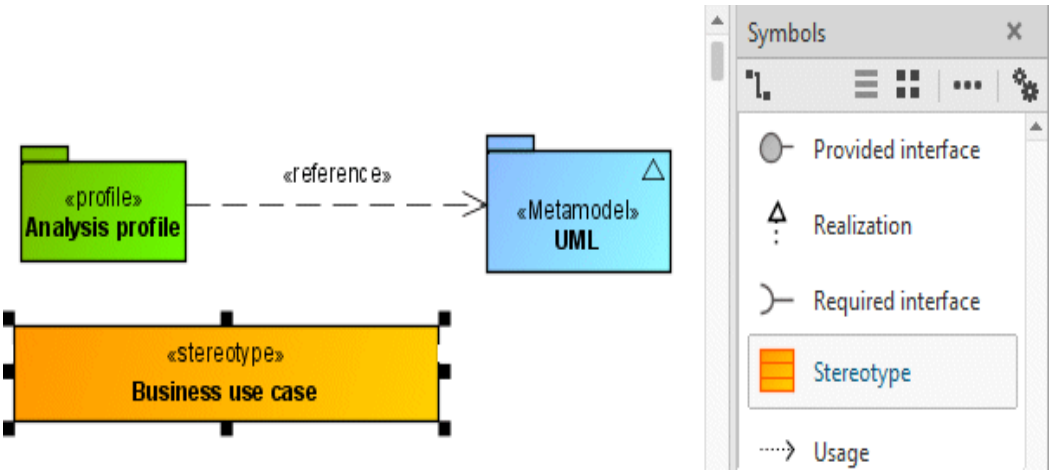


Figure 212: Definition of a new stereotype

The next step is to place the metaclass to be extended from the generated UML metamodel in the diagram and to link the stereotype with the metaclass using an Extension relationship.

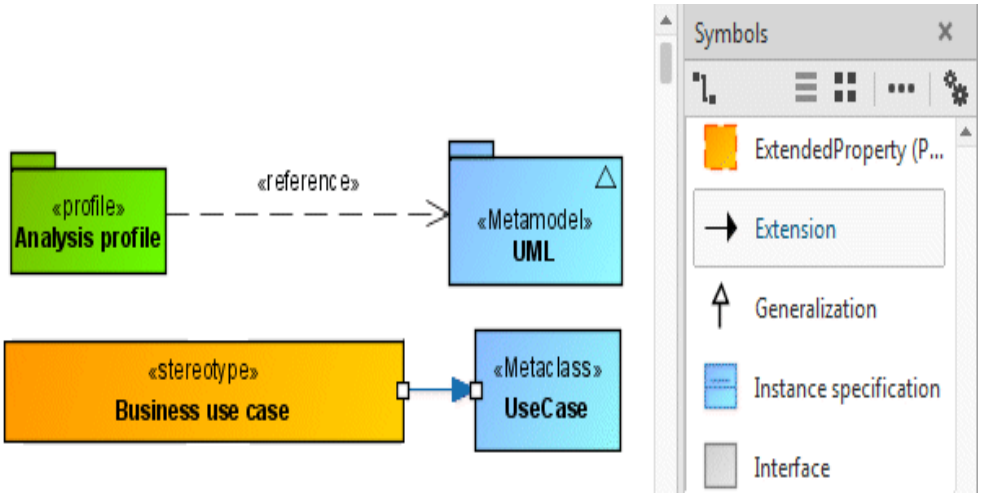


Figure 213: Linking the stereotype with the corresponding metaclass

[31] Of course, you could also create and edit the stereotype in the Explorer tree. However, the link with the corresponding metaclass in particular will then involve considerably more work than with graphical modeling.

In the example diagram, the representation options for the Extension relationship have been adjusted so that no multiplicities and names are displayed (Extension is a special form of the association), and for the metaclass the feature view of details has been suppressed so that no attributes are displayed.

For user-defined stereotypes, their description texts are displayed in the stereotype assignment dialog.

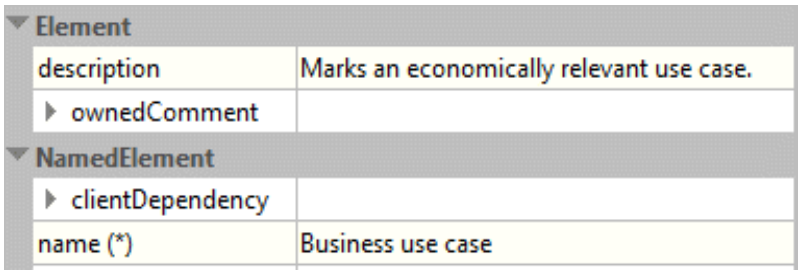


Figure 214: Description of the stereotype

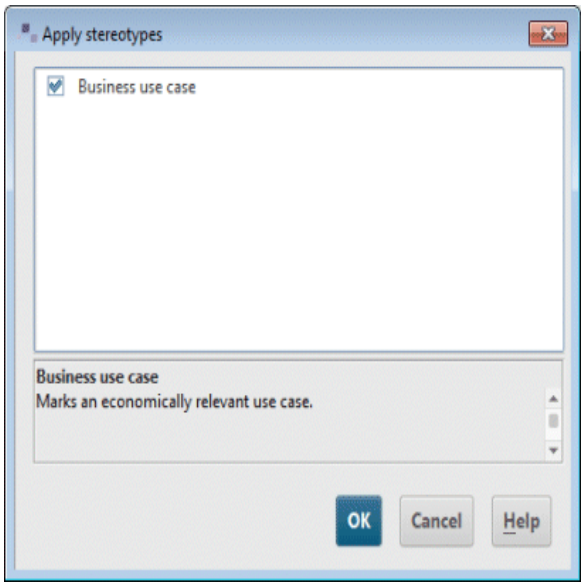


Figure 215: User-defined stereotype with description in the stereotype selection dialog

If you only want to allow use cases with the **«Business use case»** stereotype in packages to which you have assigned the Analysis profile, you can indicate this on the Extension relationship in the Profile diagram by selecting **Specify as 'required'** in the pop-up menu or clicking the corresponding button in the **Content** tab bar:

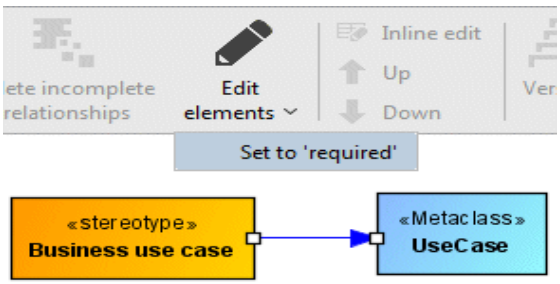


Figure 216: Flagging an Extension relationship as required in the **Content** tab bar

In the Profile diagram, the keyword **{required}** is displayed on the Extension relationship[32]:

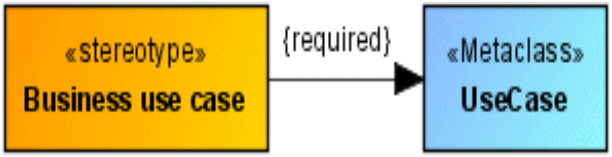


Figure 217: Extension relationship with **required** property

This change means that when you create a use case in a package to which the profile is assigned, it is automatically given the **«Business use case»** stereotype. However, setting this property only makes sense if you have not defined more than one stereotype in your profile for the corresponding metaclass.

[32] Flagging an Extension relationship as **required** changes the multiplicity of the association end linked to the stereotype from [0..1] to [1]. As the multiplicities of Extension relationships are not normally displayed in diagrams, the keyword **{required}** is displayed on the edge instead.

6.3.4 Definition of new properties

You can use stereotypes to define new properties that are not provided in the UML metamodel. In exactly the same way as the predefined properties, the user-defined properties can represent simple data values or relationships with other elements.

To define the new property **Requirements** for a stereotype, first create a new attribute of the UML type **Property** and give it the corresponding name.

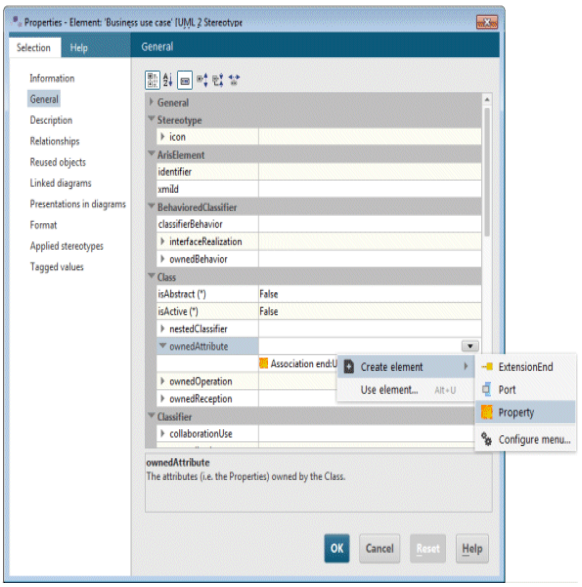


Figure 218: Creating a new attribute for a stereotype

In the next step you have to specify the type of the property. The type can be a primitive data type, an enumeration type, a metaclass, or another stereotype.

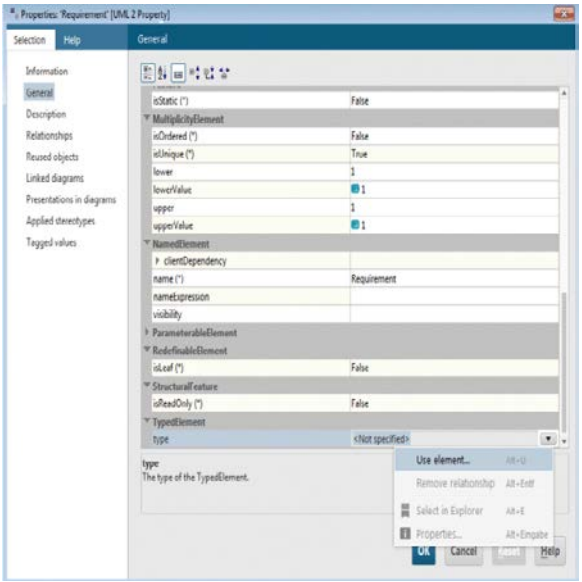


Figure 219: Specifying the attribute type

Note that only the primitive data types created by the UML metamodel generator can be used. You will find these in the **PrimitiveTypes** (Boolean, Integer, Real, String, and UnlimitedNatural) and **Base** (TimeStamp) metamodels. User-defined primitive data types are not supported, even if they have the same name as those from the generated metamodels.

The values of the **Requirements** property are entered as text. Therefore, you should select the primitive data type **String** from the **PrimitiveTypes** metamodel as the type of the attribute.

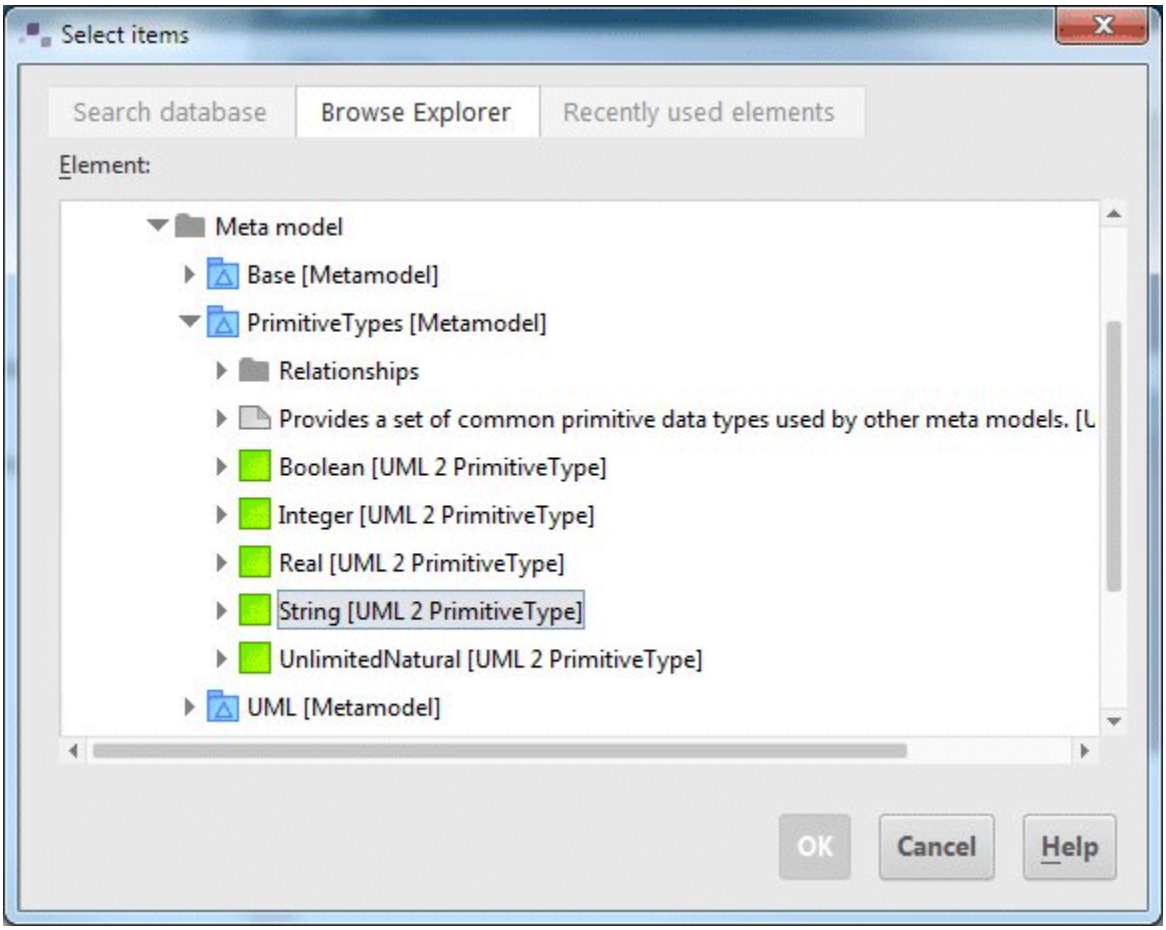


Figure 220: Selecting the primitive data type **String** from the PrimitiveTypes metamodel

The stereotype thus looks like this:

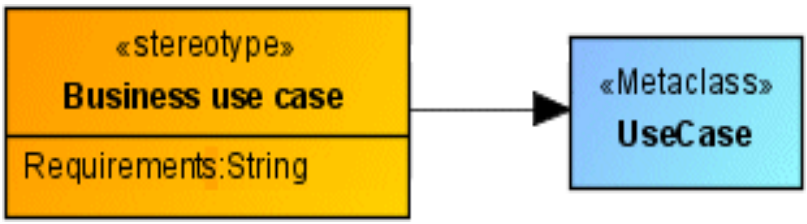
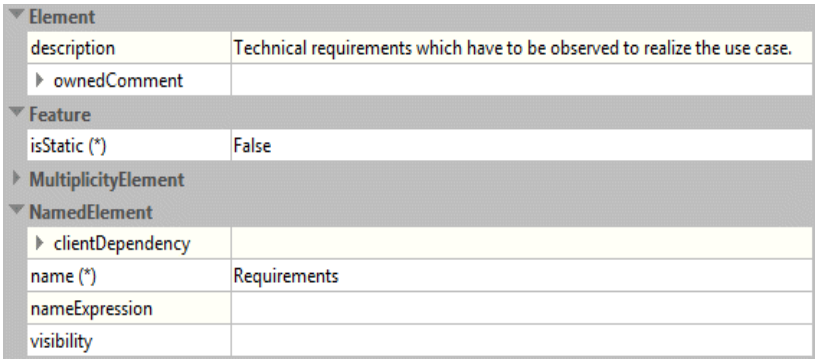


Figure 221: Stereotype with text attribute

If you enter a description for the property in addition to the name and the type, this is displayed in the Properties dialog for elements to which this stereotype is assigned.



Element	
description	Technical requirements which have to be observed to realize the use case.
ownedComment	
Feature	
isStatic (*)	False
MultiplicityElement	
NamedElement	
clientDependency	
name (*)	Requirements
nameExpression	
visibility	

Figure 222: Description of the stereotype attribute

In all use cases to which the **«Business use case»** stereotype is assigned, the new property **Requirements** is now available to you on the **Property values** properties page.

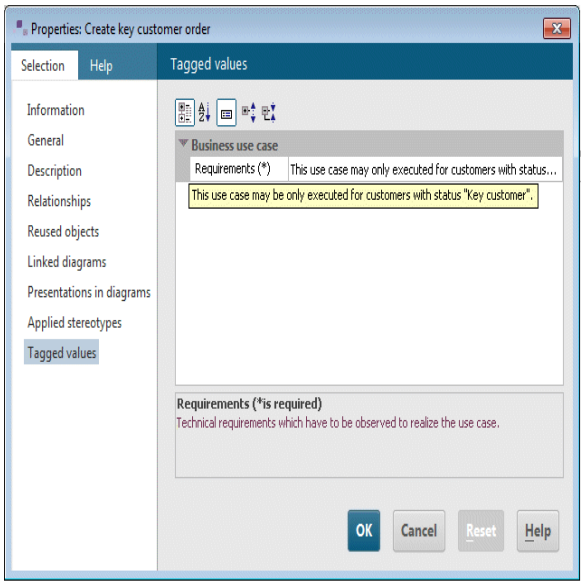


Figure 223: Property value with free text input

If you want to use an enumeration type (UML type **Enumeration**) as the type for a stereotype attribute, it must either be contained in the same profile or be imported into the profile for the stereotype from another profile using an ElementImport relationship[33].

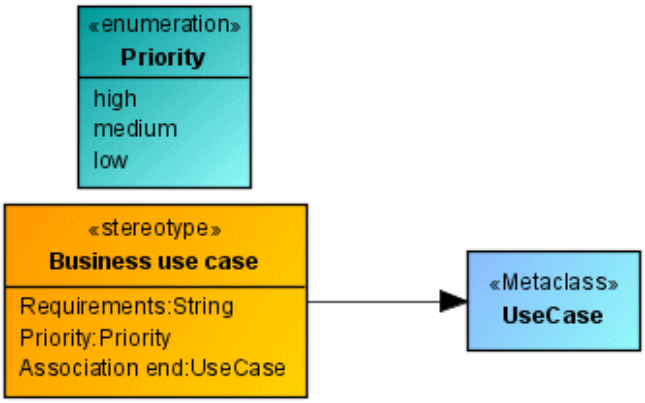


Figure 224: Definition of the **Priority** property of the **Priority** type

Figure 224 shows a UML enumeration type **Priority** with the three enumeration values **high**, **normal**, and **low** and a stereotype attribute of the same name, which uses this enumeration type as its type.

In all use cases to which the **«Business use case»** stereotype is assigned, the new property **Priority** is thus available to you on the **Property values** properties page.

[33] To import an enumeration type from another profile into the profile for the stereotype, create a presentation of the enumeration type in the Profile diagram for the stereotype and create an edge of the **Element import** type from the profile to the enumeration type.

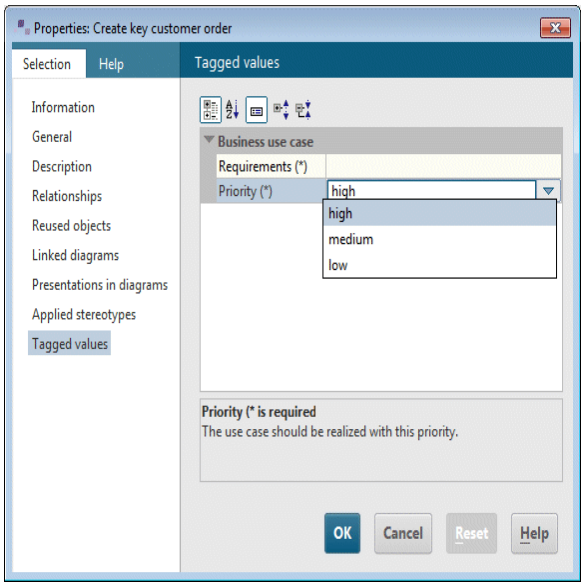


Figure 225: Property value with selection list

Properties with metaclasses as the type enable you to use elements from the corresponding metaclass as values. If you want to use a metaclass as the type for a stereotype attribute, the same applies as for primitive data types – only metaclasses that have been created by the UML metamodel generator are supported.

If you are using a stereotype as the type, you can use elements to which this stereotype is assigned as values. The stereotype must either be defined in the same profile or imported into the profile.

6.3.5 ARIS-specific features of user-defined properties

If you want to maintain the values of a user-defined property in multiple languages, i.e., depending on the database language selected at login, you must define this in the corresponding stereotype attribute. Stereotype attributes are of the UML type **Property**. As the UML specification does not recognize multilingual elements, there is no property for the UML type **Property** that you can use to specify whether or not the UML property defines a multilingual property.

ARIS-specific properties such as multiple languages are provided for definition of profiles by the OMF Extension Profile. This is a predefined profile in ARIS UML Designer, which you can use directly in exactly the same way as the default profile.

In order to be able to define multilingual properties in your profile, you must first apply the OMF Extension Profile on your profile.

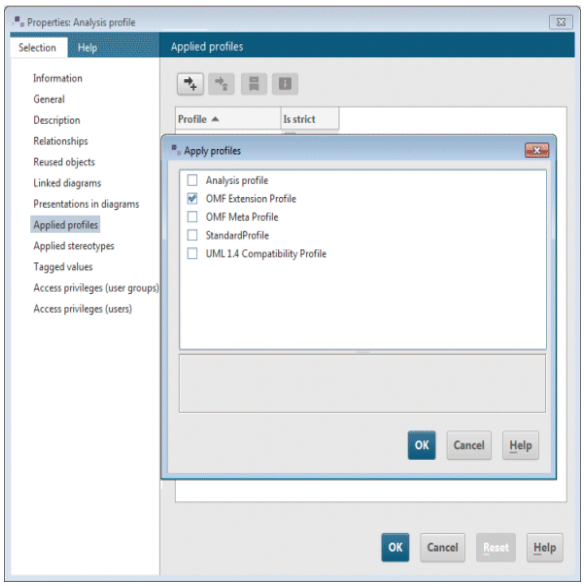


Figure 226: Assigning the predefined **OMF Extension Profile** to the user-defined **Analysis profile**

Then open the Properties dialog for the stereotype attribute whose values you want to maintain in multiple languages, and assign it the **ExtendedProperty** stereotype.

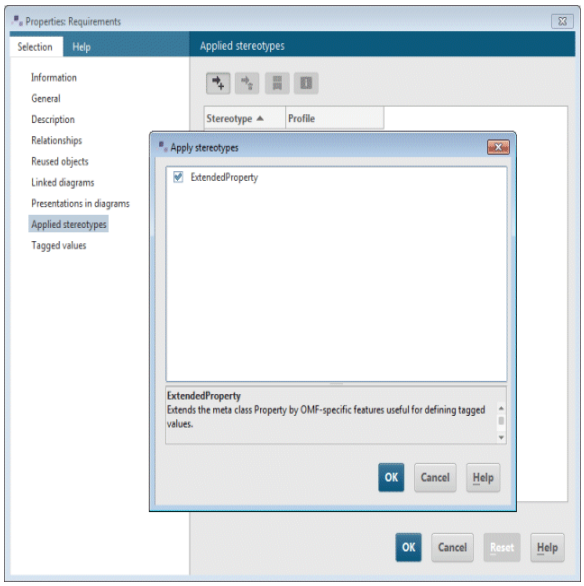


Figure 227: Assigning the **«ExtendedProperty»** stereotype to the stereotype attribute **Requirements**

The two ARIS-specific properties **isLanguageDependent** and **isStyledDocument** are then available on the **Property values** properties page for the stereotype attribute.

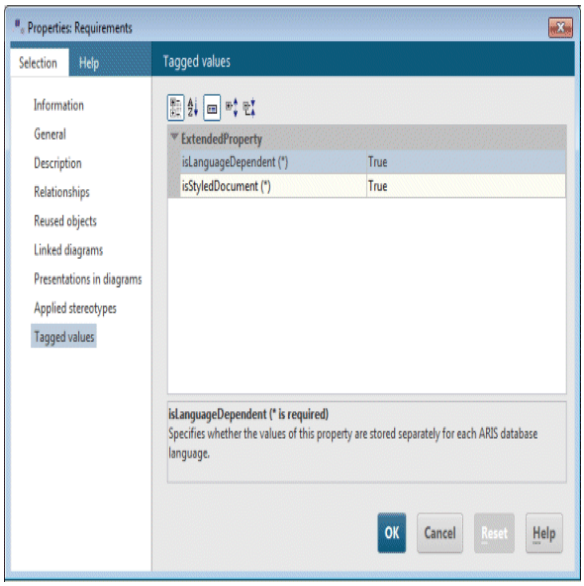


Figure 228: ARIS-specific properties of a stereotype attribute

You can use **isLanguageDependent** to specify whether property values based on this stereotype attribute are saved according to the relevant database language. You can use **isStyledDocument** to specify whether text formatting is available when editing these property values.

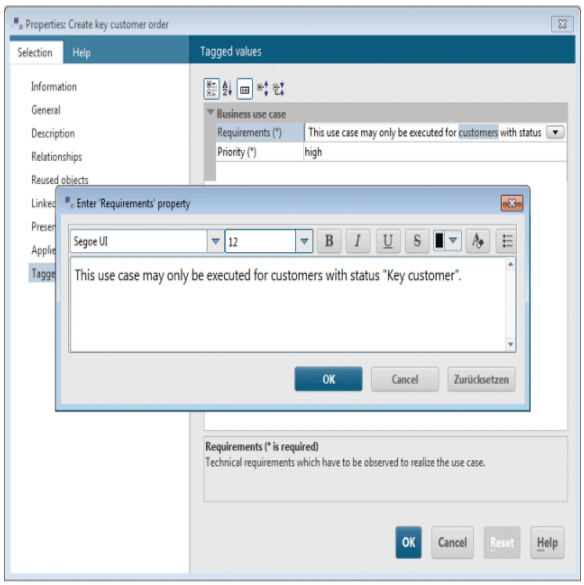


Figure 229: Editing a property value with text formatting

6.3.6 Inheritance relationships between stereotypes

If different stereotypes each define identical properties, we recommend defining a stereotype for the shared properties and having the others inherit from it.

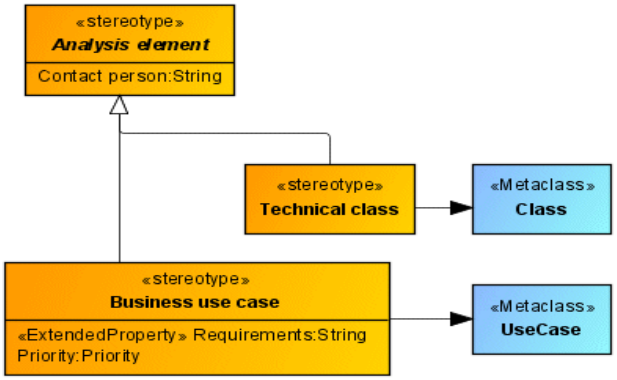


Figure 230: Inheritance relationships between stereotypes

Figure 230 extends the example from Figure 224 with the two stereotypes **«Business class»** and **Analysis element**. The **«Analysis element»** is abstract[34] and defines the **Contact** property of the **String** type. The other two stereotypes inherit from «Analysis element».

This means that all use cases with the **«Business use case»** stereotype and all classes with the **«Business class»** stereotype show the **Contact** property on their **Property values** properties page.

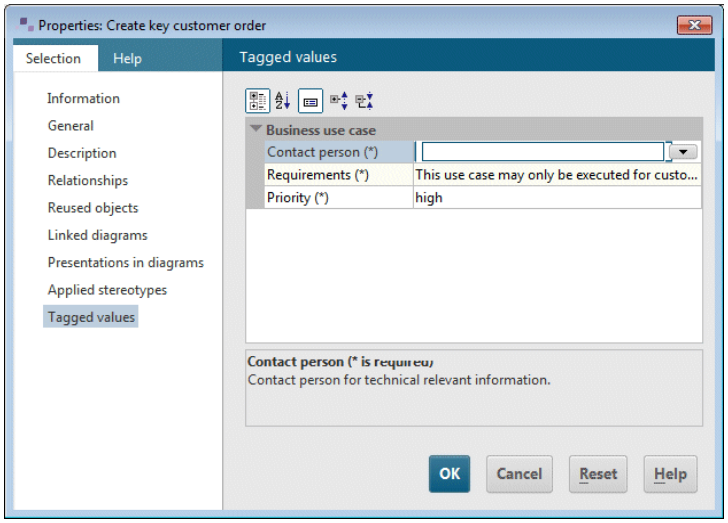


Figure 231: Property values with inherited property

The fact that the **«Analysis element»** stereotype is abstract and does not itself extend a metaclass means that it cannot be assigned to a UML element. It is merely used to define a property that is to be common to several stereotypes.

Of course, you can also create inheritance relationships between non-abstract stereotypes.

[34] The fact that the stereotype is abstract is indicated by the fact that its name is displayed in italics in the diagram.

6.3.7 Creating a filter profile

In section 6.3.2Creating a profile you have seen how to create a profile that completely supports UML by creating a relationship of the **Meta model reference** type between the profile and the UML metamodel. Within a package to which this kind of profile is assigned, all UML symbols are still available in diagrams in addition to the stereotype-based symbols.

If you delete the **Meta model reference** relationship in the profile in Figure 213[35], the **Symbols** bar for a Use case diagram only displays two symbols for the **«Business use case»** stereotype, as it is the only permitted element type.

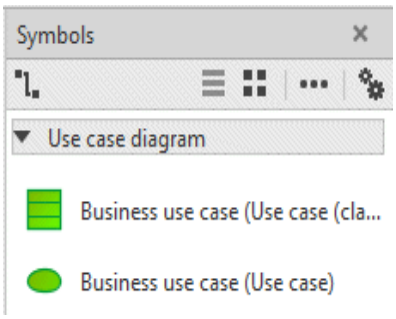


Figure 232: **Symbols** bar in the Use case diagram

The toolbars for other diagram types in which use cases are not allowed are completely empty.

You can selectively add individual metaclass to the profile by placing them in the Profile diagram and linking them to the profile using the **Meta class reference** relationship.

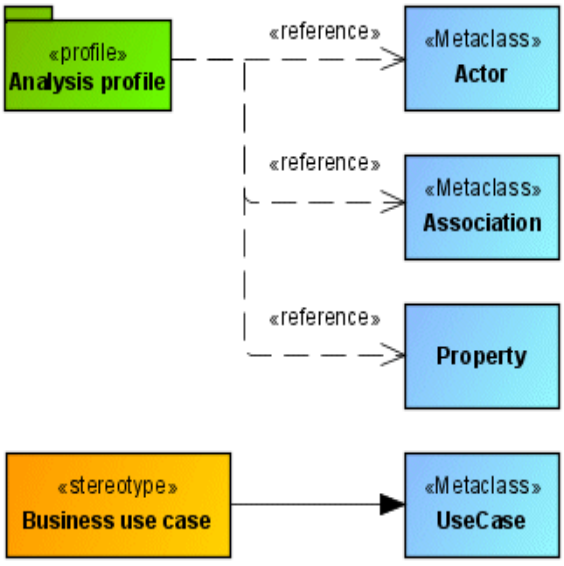


Figure 233: Profile that supports only a few metaclasses

[35] Here, **delete** means deleting the element and not just its graphic presentation in the diagram.

This profile specifies that all elements of the **Actor**, **Association**, and **Property** types can be used in diagrams, along with elements of the **UseCase** type, if the **«Business use case»** stereotype is assigned to them.

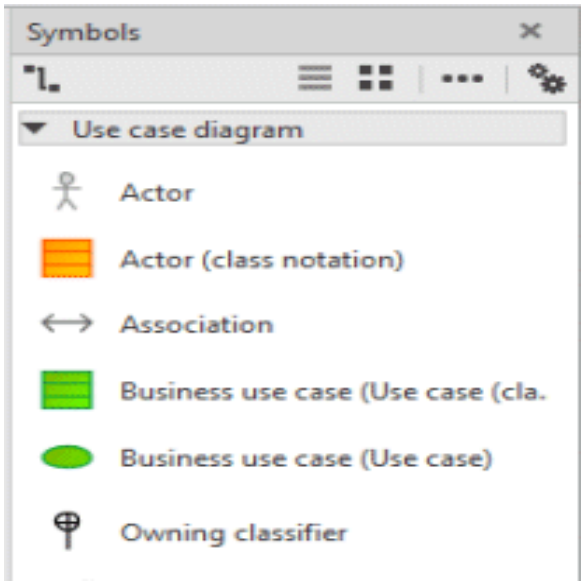


Figure 234: **Symbols** bar in a Use case diagram when using the profile

This type of profile definition requires some prior knowledge of the UML metamodel and its graphical representation in diagrams. In the above example, if the **Property** metaclass were not assigned to the profile, the **Association** symbol would not be available in diagrams, as an edge presentation for an association can only be created in conjunction with its two association ends, which are of the **Property** type.

In addition to the symbols whose metaclasses are permitted by the profile, the **Symbols** bar also displays all edge symbols that represent a meta association[36] and whose two end types are also permitted by the profile. In Figure 234 this applies to the **Owning classifier** symbol. It does not represent a metaclass, but the meta association **A_ownedUseCase_classifier**, by which a classifier can own use cases. The two metaclasses **UseCase** and **Actor** permitted in the profile inherit from the **Classifier** metaclass, which means that relationships of this type are also possible when using the profile.

Note that a profile only has an effect on the new things you can create in a diagram. Existing diagram content not supported by the profile is retained in the superior package even after assigning the profile to the diagram.

[36] You can use these symbols to create a direct relationship between two UML elements.

6.3.8 Creating a diagram stereotype

In ARIS UML Designer, you can define stereotypes for diagrams, so that they can be extended with user-defined properties, or to specify which content is to be permitted in the diagrams.

Stereotypes for diagrams are created in exactly the same way as stereotypes for elements, except that the Extension relationship is created from the stereotype to a meta diagram rather than to a metaclass. Meta diagrams are created by the metamodel generator as UML components with the **«MetaDiagram»** stereotype.

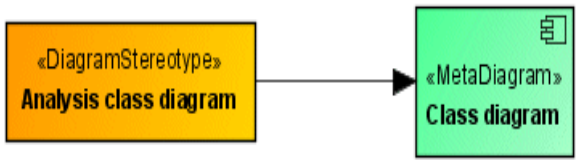


Figure 235: Definition of a stereotype for Class diagrams

New properties are defined in exactly the same way as that described for elements in section 0Definition of new properties.

In addition, you can specify which symbols can be included in the **Symbols** bar for a diagram to which the stereotype is assigned. For this purpose, you must assign the profile the OMF Extension Profile introduced in section 0ARIS-specific features of user-defined properties and assign the stereotype the **«DiagramStereotype»** stereotype.

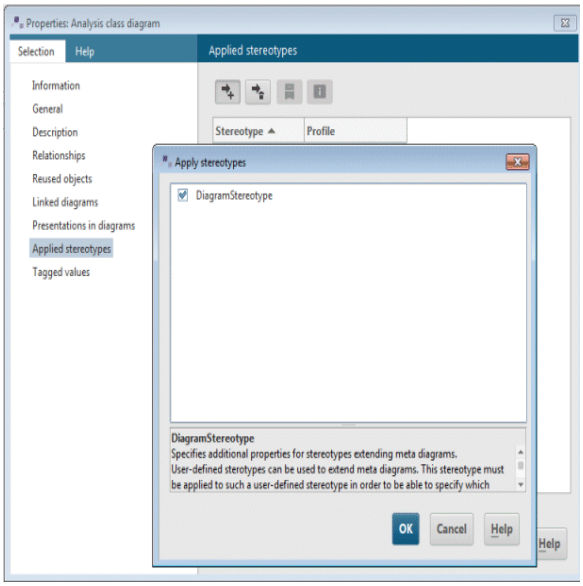


Figure 236: Assigning the predefined **«DiagramStereotype»** stereotype to the user-defined stereotype **«Analysis class diagram»**

The **«DiagramStereotype»** stereotype defines three new properties for stereotypes, which you can use to specify which symbols are permitted in the corresponding diagram:

supportedMetaClass

Here, you can add all metaclasses whose symbols are to be supported by the diagram without having to set any restrictions in terms of the symbol or stereotype to be used.

supportedStereotype

Here, you can add all stereotypes whose symbols are to be supported by the diagram. This is useful if the corresponding elements are not to be permitted in the diagram without a stereotype.

supportedSymbol

Here, you can add all symbols to be permitted in the diagram. This is useful if you only want to permit certain symbols in the diagram for a metaclass or a stereotype. As long as you do not specify any symbol in this property (as described in section 6.3.7Creating a filter profile), all edge symbols based on meta associations whose end types are supported by the profile and the other two properties are also permitted in the diagram. However, as soon as you specify a symbol here, you must add to this property all edge symbols based on meta associations that are to be permitted in the diagram.

If you do not enter a value for any of the three properties, all symbols are permitted in the diagram.

You can use the following configuration to specify that the **Symbols** bar for Class diagrams with the «Analysis class diagram» stereotype contains all symbols for the UML types Comment and Constraint, all symbols for the UML type Class with the «Business class» stereotype, and the Association, Constrained element, and Annotated element symbols.

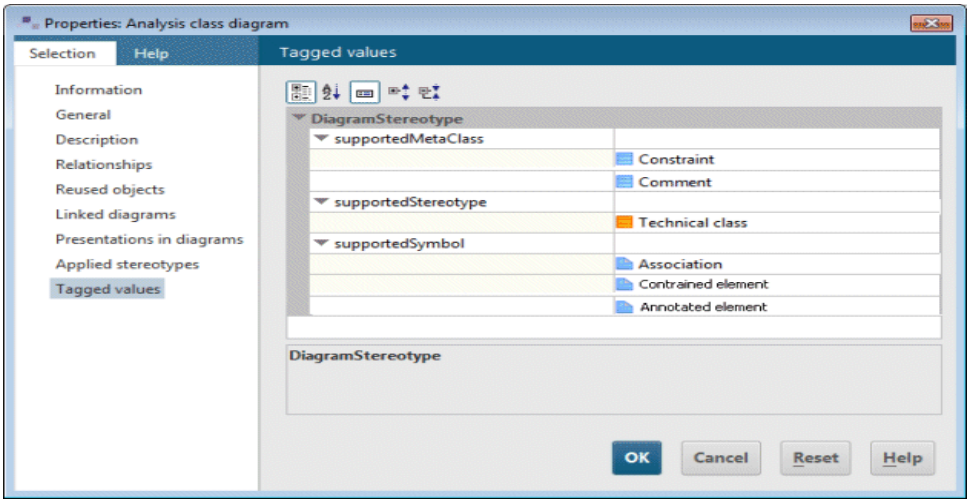


Figure 237: Configuration of a diagram stereotype

In the Profile diagram, the stereotype with this configuration is displayed as follows:

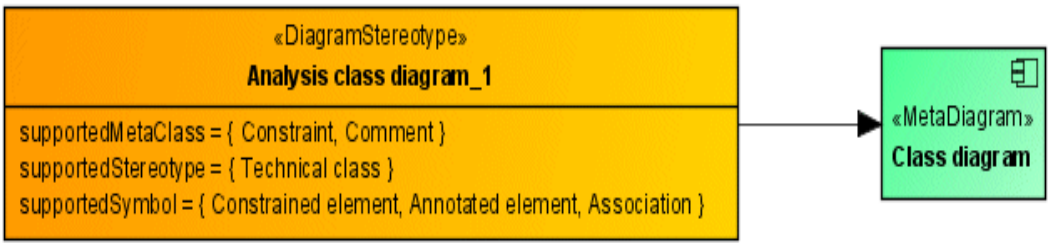


Figure 238: Configured diagram stereotype in Profile diagram

With this configuration, the **Symbols** bar for a Class diagram with the «Analysis class diagram» stereotype can contain the following symbols:

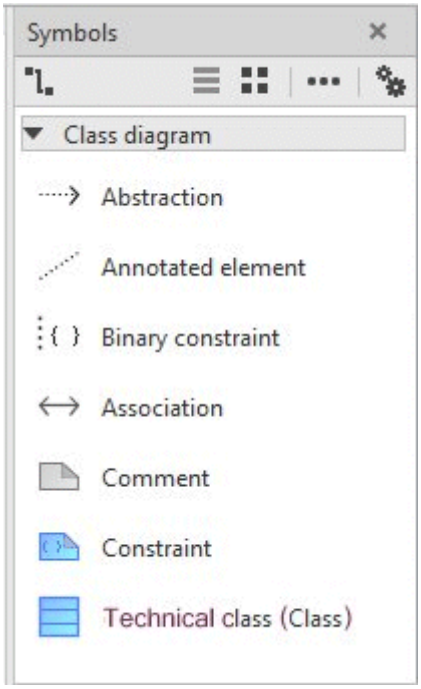


Figure 239: **Symbols** bar for an Analysis class diagram

In you have defined a single stereotype for a meta diagram in your profile and, in packages to which the profile is assigned, diagrams of this type are only to be created in conjunction with this stereotype, by flagging the Extension relationship as **required** you can specify that the stereotype is automatically assigned when creating the diagram (see also section 0Creating a stereotype).

7 Differences from ARIS Architect and ARIS Designer

This section shows you the differences in operation between ARIS UML Designer and ARIS Architect/ARIS Designer and the reasons for them.

7.1 Relevance of the model and its diagrams in terms of semantics

There is a fundamental difference between the classic ARIS Method and UML that has major effects on how you use ARIS Architect/ARIS Designer and ARIS UML Designer.

In UML, the semantics of a model (not a diagram) are completely contained in its elements and their properties and relationships. Diagrams merely represent a graphical view of the model. If you were to delete all UML 2 diagrams in an ARIS database, the semantics of the UML model would be fully retained. In addition, many UML elements are not represented graphically in diagrams and appear there in text form at most within the graphical presentation of a superior element.

In the ARIS Method, diagrams have a much greater significance. For some ARIS objects, the symbol by which they are represented graphically in a diagram actually determines their semantics[37]. As a consequence, the diagrams in the ARIS Method make a significant contribution to the semantics of the model. Conversely, objects and relationships that are not represented graphically in a diagram are irrelevant in the ARIS Method.

During a database reorganization[38] all ARIS objects and ARIS relationships with no occurrence in a diagram are therefore deleted. By contrast, UML 2 elements are retained after a database reorganization, as merely the fact that a UML element is not represented in any diagram does not reveal whether or not it is still required.

7.2 The Save and Undo/Redo functions

Unlike in the ARIS standard, in ARIS UML Designer many elements and relationships are not created and edited graphically in diagrams, but in the Explorer tree and in Properties dialogs. Therefore, the Explorer tree is far more important as a modeling component than is the case in the ARIS standard. There, the tree is primarily used for editing the group hierarchy, creating diagrams, and for navigation.

In ARIS UML Designer, changes made in the Explorer are not saved until they are saved in the database. In addition, changes can be undone provided they have not yet been saved. Only functionalities that are processed on the ARIS server require the changes to be saved immediately, e.g., copying structures in Explorer.

Apart from editing the graphic properties of an element presentation, UML elements can be edited in various components in ARIS UML Designer. For example, it is totally irrelevant whether you select a class in

[37] One example is the **Rule** object. It is the graphical representation in a diagram with a **XOR**, **AND**, **XOR/AND**, **OR/XOR** symbol etc. that specifies the exact meaning.

[38] Administrative functionality in ARIS Architect for deleting objects and connections that are no longer required.

the Explorer tree or in one of the diagrams to edit attributes, operations, or other properties on the **General** properties page.

All changes made to an element in a component of ARIS UML Designer are immediately displayed in all other components. In particular, the Explorer tree displays elements created or renamed in diagrams immediately and not only after saving, as is the case in ARIS Architect/ARIS Designer.

In addition, a change to an element in the Explorer tree or in a diagram can result in changes in other diagrams if edges or nesting relationships that appear there are rendered invalid by the change.

Because of the facts outlined, in terms of the **Save** and **Undo/Redo** functions ARIS UML Designer has very different behavior than ARIS Architect/ARIS Designer:

- The **Save** function always saves all changes you have made. Changes in the Explorer tree are also not saved until this time.
- The **Undo/Redo** functions operate globally across Explorer and diagram boundaries.

7.3 Opening diagrams

If you open a diagram in ARIS Architect/ARIS Designer, it is locked for write access by other users when opened, or a message is displayed stating that it can only be opened in read-only mode if another user has already opened the diagram.

To prevent a diagram being automatically locked for changes by other users when you only want to view it, in ARIS UML Designer diagrams are always opened in read-only mode. The corresponding lock is only requested from the server and the diagram is updated when you attempt to change the diagram, so that you are editing the diagram in its current state. If it is already being edited by another user, you will see a corresponding message.

When saving, in ARIS UML Designer all locks are canceled.

7.4 Element hierarchies

While hierarchies of objects can only be represented in ARIS Architect/ARIS Designer graphically using corresponding connection types in diagrams or by assigning a diagram with subordinate objects to a superior object, in ARIS UML Designer element hierarchies can be represented directly in the Explorer tree, where the superior element owns the subordinate elements.

7.5 Graphical connections and edges in diagrams

In ARIS Architect/ARIS Designer, a graphical connection in a diagram always displays a single connection definition. In ARIS UML Designer, a graphical edge can represent a whole series of elements and relationships. You will find more detailed information about this issue in section 4.2Complexity of edge presentations.

7.6 Assignments

The ARIS Method specifies which and how many diagram types can be assigned for each object type. ARIS UML Designer uses a a different approach for diagram assignments, which is outlined in more detail in section 3.2.2.4Linked diagrams (elements).

7.7 Creating ARIS scripts

ARIS UML Designer provides special functionalities for editing of UML content in ARIS reports and macros. These are outlined using some example scripts in the report and macro categories under **UML example scripts**.

8 Differences from ARIS UML Designer 7.x

With ARIS UML Designer 9, both a new approach to mapping from UML to ARIS and a new approach to integration of UML with classic ARIS modeling has been introduced. This results in several changes compared to ARIS UML Designer 7.x, which are explained below.

8.1 UML version

The most obvious difference relates to the supported UML version. ARIS UML Designer 7.x supports UML 1.4, and ARIS UML Designer 9 supports UML version 2.5.

ARIS UML Designer 9 supports all element types and properties included in the UML 2.5 specification. Apart from Timing diagrams, for which there is only rudimentary support, all UML 2.5 diagram types are supported.

Thus, the UML support in ARIS UML Designer 9 is significantly more comprehensive than that in ARIS UML Designer 7.x.

8.2 Mapping of UML to ARIS

Mapping of UML 1.4 to ARIS Method represented a mixture of UML and business process types. In some cases, the decision on whether a UML element type was mapped to an ARIS object type or ARIS connection type was based on how it was to be displayed graphically in diagrams. Thus, the reusability of business process objects as UML elements was specified in the ARIS Method.

Element hierarchies were realized using ARIS connections between the corresponding objects. For example, the nesting of a class in a package was mapped using a connection of the **CT_IS_NESTED** type. This could mean that a class was contained in various packages due to multiple connections of this type, which is not allowed in UML.

As some UML elements themselves were mapped to connections, e.g., Dependency, ARIS UML Designer 7.x internally supported edges as the source or target of connections.

Mapping of UML 2.5 to the ARIS Method involved a new approach, which is outlined in section 4Mapping UML to the ARIS object model. For each UML element type, a single corresponding UML 2 object type exists in the ARIS Method. All UML elements are mapped to ARIS groups with the corresponding object type number.

These changes also mean that databases with UML 1.4 content created using ARIS UML Designer 7.x have to be converted to UML 2 before this content can be displayed and edited by ARIS UML 9. Further information can be found in section 0Data transfers from ARIS UML Designer 7.x.

8.3 Reuse of business process objects in UML

The concept of direct reuse of some specifically defined ARIS object types in UML that is familiar from ARIS UML Designer 7.x has been replaced with a flexible new concept in ARIS UML Designer 9, which means that

as a user you can now decide which business process objects you want to map to which UML elements. Direct reuse has been replaced by a new reuse relationship for this purpose.

This concept is outlined in detail in section 5Linking business process and UML modeling.

8.4 Saving and undoing changes

In ARIS UML Designer 7.x, diagrams were individually saved but all changed UML elements were always saved when saving an individual diagram, i.e., including the elements that were changed from other diagrams. Changes in the UML package tree were always saved directly, which meant that they could not be undone.

ARIS UML Designer 9 always saves all changes. The **Undo/Redo** functionality operates globally across all components and diagrams.

Section 7.2The Save and Undo/Redo functions describes the motivation behind this changed behavior in ARIS UML Designer 9.

8.5 Integration of UML into the Explorer tree

In addition to the Explorer tree, ARIS UML Designer 7.x contained a second tree – the UML package tree – for UML-compliant display and management of UML content and hierarchies. For each UML package, UML model, and UML profile both an ARIS group and an ARIS object were created to represent the UML package hierarchy in the Explorer tree in the form of an ARIS group hierarchy and to enable all UML elements contained in the package to be stored in the corresponding group for the package.

Particularly when working in the Explorer tree, it was possible that the UML structure of the two trees was no longer synchronized and they were showing different UML hierarchies.

In ARIS UML Designer 9, UML has been fully integrated into the Explorer tree. In addition, UML elements are mapped to groups with the corresponding object type number, which means that there are no longer two different ARIS items (group + object) for a UML package.

8.6 Separate window for ARIS UML Designer

The functionalities and components of ARIS UML Designer 7.x were completely integrated into the ARIS Architect or ARIS Designer window.

Because of the different concepts, particularly in terms of the Explorer tree, the Save and Undo behavior of ARIS UML Designer 9 and ARIS Architect/ARIS Designer, separation of the two applications using two different windows is now necessary.

However, the two are closely integrated on a functional level, which means that easy navigation from one application to the other is possible. You can find more detailed information about this topic in section 5.3Navigation between <aba>/ARIS Designer and ARIS UML Designer.

8.7 XMI interface

ARIS UML Designer 7.x supported the XMI format UML 1.4/XMI 1.1, while ARIS UML Designer 9 supports UML 2.5/XMI 2.1.

In contrast to ARIS UML Designer 7.x, the XMI import in ARIS UML Designer 9 identifies content of the XMI file to be imported that already exists in ARIS and does not duplicate it.

In addition, the XMI interface in ARIS UML Designer 9 also supports export and import of diagram information based on the UML Diagram Interchange Standard.

You can use XSL transformations, which you manage in the Administration component on the ARIS server, you can make individual adjustments to external formats both for XMI export and for XMI import. Further information can be found in section 0XMI resources.

9 Appendix

9.1 Glossary

Booch method

(Object-oriented modeling language developed by Grady Booch, a forerunner of UML)

BPMN (Business Process Model and Notation)

(Modeling language for business processes, standardized by the OMG)

MOF (Meta Object Facility)

(Architecture for metamodels and their implementation, standardized by the OMG)

OMF – Object Modeling Framework

(ARIS implementation of MOF, part of the ARIS UML Designer architecture)

OMG – Object Management Group

(Non-profit organization, publishes standards for the IT industry)

OMT – Object Modeling Technique

(Object-oriented modeling language developed by James Rumbaugh et al., a forerunner of UML)

OOSE– Object-Oriented Software Engineering

(Object-oriented modeling language developed by Ivar Jacobson, a forerunner of UML)

UML – Unified Modeling Language

(Object-oriented modeling language, standardized by the OMG)

W3C – World Wide Web Consortium

(Committee for standardization of technologies in the World Wide Web)

XMI – XML Metadata Interchange

(XML format for metamodels and models based on them, standardized by the OMG)

XML – Extensible Markup Language

(Language for mapping of hierarchical structures in text form, standardized by W3C)

XSL – Extensible Stylesheet Language

(Language family for definition of the layout of XML documents, standardized by W3C)

XSLT – XSL Transformation

(Transformation language for transferring documents from one XML format to another, standardized by W3C)

9.2 Additional documents and references

9.2.1 Documents

UML Migration Guidelines – Guidelines for migration of an ARIS 7.x database with UML 1.4 content to UML 2.5

9.2.2 References

OMG(Object Management Group)**:**www.omg.org

OMG specifications (BPMN, MOF, UML, XMI etc.): www.omg.org/spec

W3C (World Wide Web Consortium): www.w3.org

W3C specifications (XML, XSLT etc): www.w3.org/standards