SIS - Entry Form User’s Manual

Version 2.3

Institute of Computer Science

Foundation for Research and Technology - Hellas

TABLE OF CONTENTS

1. Introduction

Entry Form (EF) is a tool used for the interactive update of the Semantic Index System (SIS) [CD]. SIS is a semantic network information management system. It employs an object-oriented data model based on the Telos knowledge representation language [MBJK90]. A brief introduction of the features of the Telos language as implemented in the SIS information bases follows.

The objects in a SIS information base are distinguished into *nodes* (or *individuals*) and *attributes*. The nodes represent entities that can exist independently. The attributes represent binary relationships between objects and their existence depends on the objects they connect. Attributes are directed links from the *from-object* to the *to-object*. Both nodes and attributes can (under appropriate conditions) participate in instantiation and specialization relations. There are objects that cannot have instances; these are called *Tokens*. The objects that can have instances are called *classes*. The Tokens can be classified to classes, the classes to meta-classes, the meta-classes to meta-meta-classes etc.

Every object of the information base may have an external name. The name of a node is mandatory and should be unique in the information base. This means that the user cannot re-use a name given to a node of the information base. The name of an attribute is optional and should be unique in the scope of the from-object. This means that the user cannot re-use a name given to an attribute, which is already assigned to the from-object. In the rest of this manual, an attribute will be indicated by its full name, i.e. the name of the attribute and the name of the from-object in brackets (e.g. *wheel (Car)* is the full name of the attribute *wheel* which is assigned to the node *Car*).

The update of the information base can be achieved with operations on objects. The available operations are creation, deletion, renaming, classification (assignment to a class), generalization (assignment to a superclass) and attribute assignment. EF provides the user with the capability to perform the whole set of the prementioned operations.

The user updates the information base through EF in a task-oriented way. This means that the user can update the information base through tasks, which have been assigned to him/her. A task is defined by the objects that are allowed to be updated and the operations that can be performed on these objects. In this way, parts of the information base can be isolated and updated independently from others, using predefined operations.

EF is based on a three-activity process model [DT95], [Das96a], whose flow chart is shown in Figure 1.

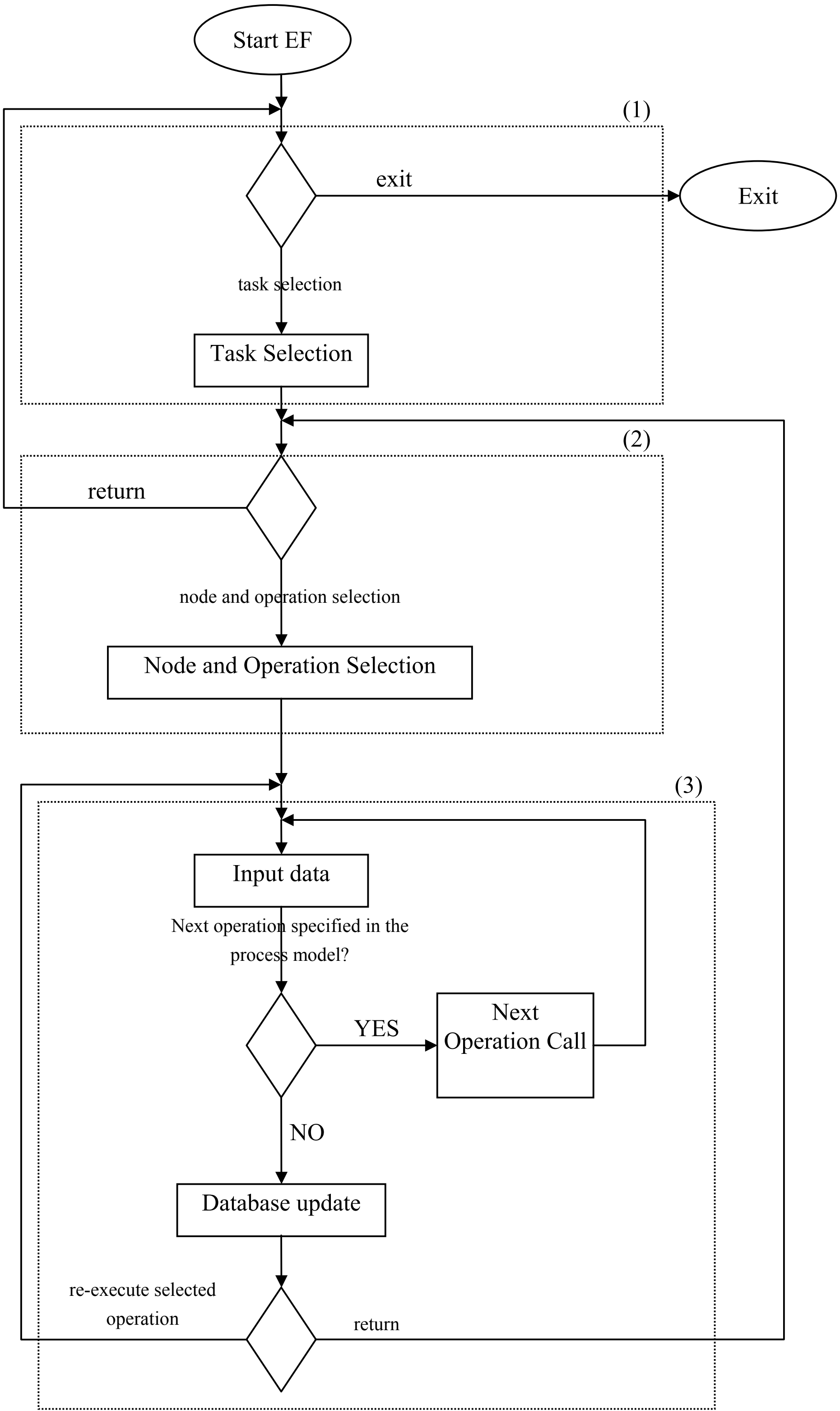

Figure 1 Three-activity process model

The activities (task selection, node and operation selection, update of the information base) are shown in the frames of the flowchart, with labels (1), (2) and (3).

The first activity is the task selection: the user selects the task through which (s)he wants to update the information base.

- The second activity is the node and operation selection: the user selects nodes [1]and operations available in the selected task.
- The third activity is the actual update of the information base: having selected the update operation, the user inputs data, which update the selected node. Depending on the specifications given in the process model for the selected operation, a sequence of operations that follow the selected one may be created. The operations are performed on the selected node, while the last operation of the sequence causes the update of the information base with all the changes of the operations.

EF provides the user with a set of forms through which each activity is performed. There is the task selection for the first activity, the node and operation form for the second activity and six update forms for the third activity: forms that implement operations for creation, deletion, renaming, classification, generalization and attribution for nodes and attributes. Each of these forms can be customized by the process model, in order to meet the application's requirements.

Task assignment and operation specification is governed by the process model of the EF [Das96a]. Additionally the tool has the ability to preserve a set of integrity constraints not imposed by the semantics of the Telos language [DKT95]. This is accomplished by the use of a constraint model, which has been developed. Furthermore EF provides acceleration facilities and guiding mechanisms during the interactive update.

By using the process and the constraint model, graphical interface elements of EF such as buttons, labels, list elements can be easily customized. An emphatic large font, in the rest of this manual indicates these elements. The fixed graphical interface elements of the tool (i.e. these that cannot be customized through the process and the constraint model) are indicated by a **bold** font.

We also have to note that clicking the *left* mouse button on them does the selection of the graphical interface elements, unless explicitly stated otherwise. For example, whenever we mention that the user has to "select" or "press" a button of EF, we mean that the user has to click the *left* mouse button on it. However there are cases the user has to click the *right* mouse button on a graphical interface element. In these cases, we state it explicitly.

2. Getting started

The user can start the tool by typing:

<ef_bin_name> *<ModelName>* [*<UserPermissions>*] [*<ThesaurusName>*] [*<-Uuser name>*]

---

[1] The attributes can be selected starting from the node to which they have been assigned.

(e.g. *$SIS/bin/ef CLIO* or *$SIS/bin/ef CLIO ExpertUser TMS -Usomeone*)

The parameter <ModelName> is mandatory. It denotes the information base model name under update. The parameter <UserPermissions> is optional. It is used in case more than one user is specified in the process model [Das96a]. The parameter <ThesaurusName> is optional. It denotes a specific thesaurus name under update. The parameter <-Uuser name> is optional. It denotes that any data base change (e.g. creation, renaming, (de)classifying, moving to Hierarchy, creating/deleting an attribute of a HierarchyTerm) is marked with appropriate links pointing from target HierarchyTerm to given user (as Editor). In case this parameter is passed as "-U?", the application uses as user name, the name of the user currently logged onto the system.

EF can also be called as an external tool by GAIN ExternalTools [The95]. The latter is the tool used for navigating through the information base. The user has to select the appropriate choice on GAIN's Admin menu [DKP95]. In this case the parameters for the EF start up are specified in the setup model of the SIS external tools [The95].

After setting the appropriate parameters, EF presents its first form, the one for task selection.

3. Task form

EF allows to update the information base through predefined tasks. The user select is done by clicking the *left* mouse button on the TaskList button, located at the top of the task form (see figure 2).

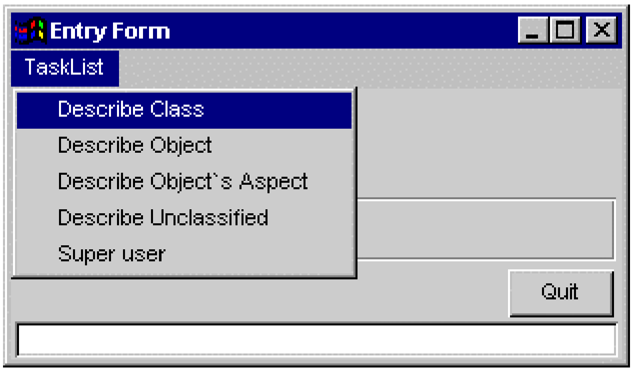

Figure 2: Task selection

The tasks are offered as choices of a pulldown menu, which appears whenever the user presses the **TaskList** button. In the example shown there are four tasks, namely DescribeClass, DescribeObject, DescribeUnclassified, and DefineRootClass.

A pulldown menu appears with the available tasks. Depending on the <UserPermissions> given as parameter on startup and the specifications given in the process model, there may exist much different set of tasks for the same information base. The user can select the task through which (s)he wants to update the information base. The selected one appears as the **Current Task** below the **TaskList** button. (See figure 3).

There are three cases with regard to the nodes that a task can update (called hereafter *object set* of the task):

- the object set is the entire information base
- the object set is a predefined part of the information base, which cannot be modified by the user
- the object set is a partially predefined part of the information base, whose complete definition depends on a user-given parameter.

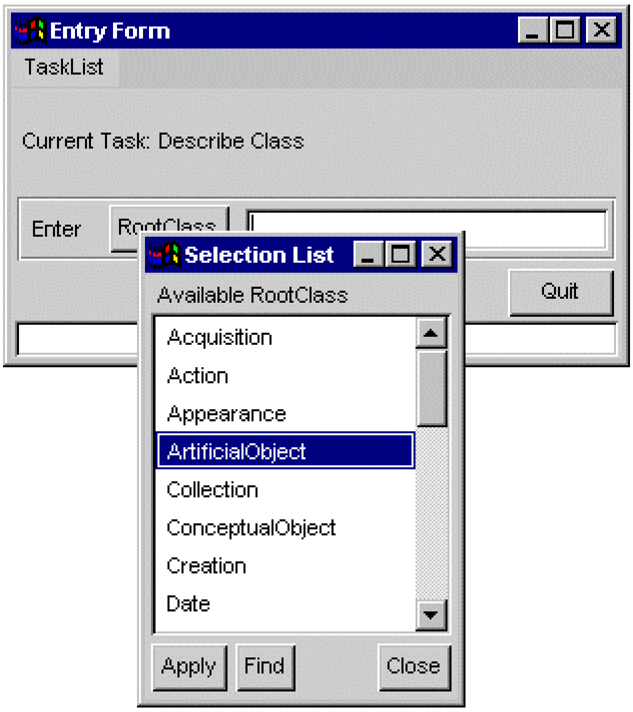

Figure 3: Argument Selection

DescribeObject is the current task. It needs an argument to be fully defined. The argument's type is *RootClass*, as indicated by the RootClass button. By pressing this button, the available arguments appear in a list. In the example shown, the argument *Artificial Object* is selected. By pressing the **APPLY** button the user provides the task with the selected argument. The same can be done by double-clicking on an available argument appears in the list

All of these cases are specified in the process model.

In the last case, after the task selection, the user has to supply the parameter that will define the task's object set. This parameter is an argument that the user gives to the task. The user has to type the argument in the text field provided for this reason and press the RETURN key on the keyboard. The argument's type is specified in the process model and it is indicated by the name of the pushbutton next to the text field (see figure 3). Alternatively, the user can press the pushbutton to get a list of the possible arguments and select one from this list (see figure 3). A pattern match searching mechanism, activated by pressing the FIND button can search the contents of the list.

4. Node and operation form

After the task selection, the node and operation form appears (see figure 4).

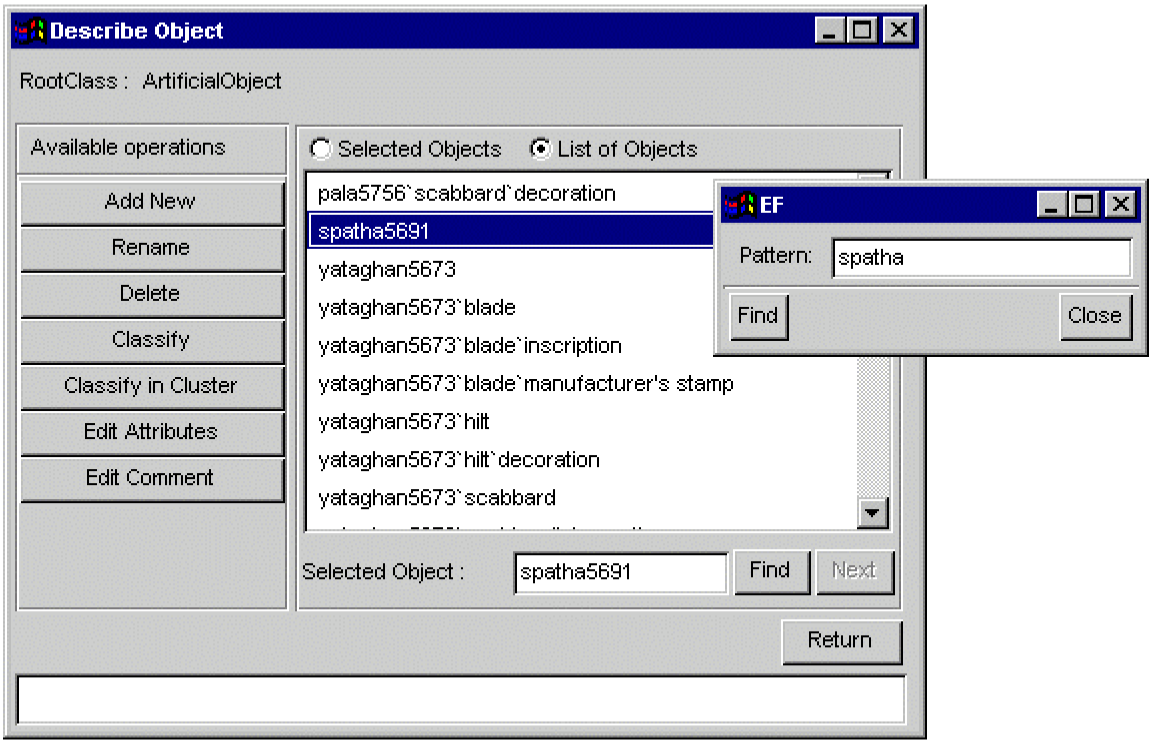

Figure 4: Node and operation selection

The task Describe Object has as argument the *RootClass ArtificialObject*. The node *spatha5691* is selected from the **List of Objects** and is displayed in the **Selected Object** text field. The pattern search mechanism, activated by using the **FIND** button, helps the user to find a node in the scrolling list. After the node selection the user can select an operation. The available operations for the task DescribeObject are AddNew, Rename, Delete, Classify, ClassifyinCluster, EditAttributes and EditComment, found in the **Available operations** area.

The task appears as the name of the form's window, while the argument of the task (if any) is displayed at the top of the form. The user can select the node that (s)he wants to update from the object set of the task. The contents of this set are displayed in a scrolling list, at the right part of the form. This list is appeared on demand, when the button *List of Objects* is pressed. By pressing the button *Selected Objects* the scrolling list displays the most recently selected nodes for updating. The user can select a node from this list. The selected node is displayed in the text field with the label **Selected Object**, below the scrolling list. When the object set is large (over 1000 elements), its contents are displayed in fixed size pages. In this case the **NEXT** button is enabled and is used to display the next page. There also exists a pattern match searching mechanism, activated by using the **FIND** button. This mechanism is applied on the whole list - not only on the currently displayed page.

On rare occasions the task may have an extremely large object set (over 100000 elements). In this case, it makes no sense to display the whole object set in the scrolling list. The user has to type the node that (s)he wants to update in the **Selected**

Object text field, and the system checks if the typed node exists in the object set. The same actions should be followed in case the task is programmed to update every object of the information base. Once again, it makes no sense to display the entire information base in the scrolling list. The user has to type the node that (s)he wants to update in the **Selected Object** text field, and the system checks if the typed node exists in the information base.

The node selection is followed by the operation selection [2]. The available operations in each task are provided to the user as a set of pushbuttons. The user can press an operation button in order to update the selected object, via the operation's update form. After pressing an operation button, an update form appears.

5. Update forms

There exist seven types of update forms corresponding to the update operation types: creation, deletion, renaming, instantiation, generalization attribute assignment and user defined operations. Each operation of the selected task has a specific update type, so it is implemented through the respective update form. The forms for node updating can be called implicitly from the node and operation form. The update of attributes can be done by means of the update attributes form (see section 5.6) of the object to which these attributes are assigned.

Although each type of update form is different from the others, as far as the graphical and the functional point of view are concerned, there are some common parts in all of them:

The form's window has the name of the operation, which it implements. See for example the forms for the operations AddNew and Delete of figure 4, in figures 5 and 6 respectively.

At the top of the form there is a label indicating the task to which the operation belongs and the argument of the task, if it exists. For example, the operations Add New in figure 5 and Delete in figure 6 belong to the DescribeObject task with argument the RootClass *ArtificialObject*, while the operation Rename in figure 7 belongs to the DescribeClass task with argument the RootClass *Action*.

Each form also has an object called *target*; it is the object that is going to be updated. The target is displayed below the task indication label. Its appearance is indicated by a label which depends on the form's type (see for example in sections 5.1, 5.2, 5.3, 5.4, 5.5, 5.6).

At the bottom of each form there is a message area and above it a **COMMIT** and a **RETURN** button. By pressing the **RETURN** button, the user can return to the previous form (the one that caused the appearance of the current form). By pressing the **COMMIT** button, the user demands the update of the information base.

In some cases, defined in the process model, a **CONTINUE** button appears in the place of the **COMMIT** button. This indicates that another one should follow the current form. By pressing the **CONTINUE** button, an update form that follows the

[2] Whenever the user wants to create a new node in the information base, the node selection is omitted.

current one appears. In this way a chain of update forms is created. The last form of the chain has a **COMMIT** button. By pressing this button the information base is updated with all the changes of the chain's forms in one transaction.

If no error occurs during the update, a message is displayed in the message area of each form, denoting successful termination. If an error occurs, EF informs the user with a message in an error message dialog box. Simple warnings, whenever needed, appear in a warning message dialog box.

Form customization Each update form is the implementation of an operation, defined in the EF process model. The operation may have a constraint set, whose semantics are given by the operation's update type [DT95], [Das96a]. This set is defined in the EF process model and it is a basic factor for the operation's functionality. In general, the constraint set is used to specialize the functionality of the operation in order to meet the needs of the task. At the same time it imposes constraints on the operation preventing undesirable updates.

The constraint set is not directly visible to the user. However, its existence affects the user's interaction with the system, while it is used to customize the operation's functionality hence the corresponding form's operability.

The absence of the constraint set denotes that the operation has no constraints. Such operations allow the user to execute all the primitive updates available for the operation's update type, provided that the Telos constraints [DTK95] are satisfied.

After these remarks, we present the forms, which are used by the system for the information base update.

5.1 Create Node

Nodes can be created using the form for node creation (see figure 5)

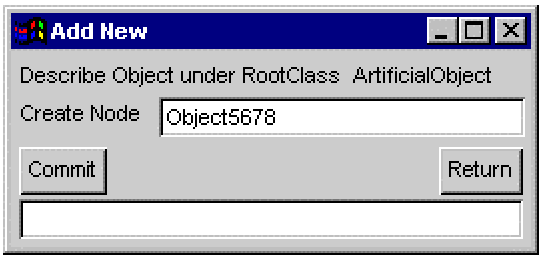

Figure 5: Create Node

Add New implements an operation for node creation. The node *Object5678* is created, when the **COMMIT** button is pressed.

The user has to type the name of the node that (s)he wants to create; this should be unique in the information base. A text field with the label **Create Node** is offered for this reason. The typed node is the target of the operation. The target will become member of the classes that exist in the constraint set of the operation. The **COMMIT** button causes the insertion of the new node in the information base.

If there is no constraint set defined in the process model, the user has to type the class or just the instantiation level [3] of the node that is going to be created. A dialog box with a text field is provided for this reason. After the specification of the node's class, the new node is inserted in the information base.

5.2 Delete Node

Nodes can be deleted using the form for node deletion (see figure 6)

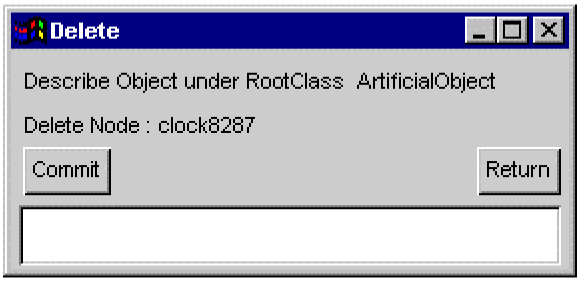

Figure 6: Delete Node

Delete Node implements an operation for node deletion. The node *clock8287* is deleted from the information base, when the **COMMIT** button is pressed.

The target node is indicated by the label **Delete Node**. By pressing the **COMMIT** button the user demands the deletion of the target. If the constraint set is defined in the process model, it is used to check whether the target can be deleted. If the deletion is allowed by the constraint set and the telos constraints, then it actually takes place; otherwise a message dialog box informs the user about the error which has occurred.

Deletion may lead to other indirect operations in order to preserve integrity constraints that may hold in the information base:

- If the target has subclasses and superclasses, the generalization links are preserved among these. This means that the target's subclasses become subclasses of the target's superclasses.
- If the target has attributes pointing to primitive values, these are deleted too.
- If the target has necessary attributes (see [Das96a]), these are deleted too.
- If the target has attributes with generalization properties (see [Das96a]), these are preserved after the deletion.

[3] Token, S_Class, M1_Class, M2_Class, M3_Class, M4_Class

5.3 Rename Node

Nodes can be renamed using the form for node renaming (see figure 7)

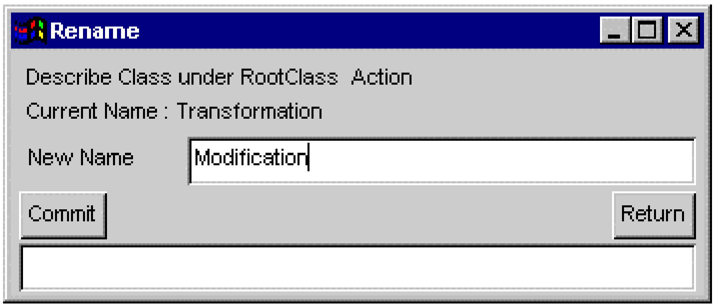

Figure 7: Rename Node

Rename Node implements an operation for node renaming. The node *Transformation* (**Current Name**) is renamed to Modification (**New Name**), when the **COMMIT** button is pressed.

The target node is indicated by the label **Current Name**. The user has to type the new name of the target in the text field with the label **New Name**. The new name should be unique in the information base. By pressing the **COMMIT** button the user demands the renaming of the target. If the constraint set is defined in the process model, it is used to check whether the target can be renamed. If renaming is allowed by the constraint set and the telos constraints, then it actually takes place; otherwise a message dialog box informs the user about the error which has occurred.

5.4 Classify Node

Classification links from the target node to the classes that the operation allows can be created or deleted using the form for node classification (see figure 8). The target node is indicated by the label Target. The classes of the target are displayed in a list, as shown in figure 8.

The links to the classes that are allowed to be deleted are displayed in <existing> state. The user demands the deletion of a classification link by pressing the toggle button <existing>, switching it to <*to be deleted*>. The states <existing> and <*to be deleted*> are complementary and can be interchanged on demand.

Whenever a classification link is going to be added to the target node, it is displayed in <*to be added*> state. The user can cancel the addition of a to be added classification link, by pressing the toggle button <*to be added*>, switching it to <*cancel addition*>. The states <*to be added*> and <*cancel addition*> are complementary and can be interchanged on demand.

- If there exists a constraint set (set of classes), then only the links to the classes of the target found in this set, can be deleted. The links to the classes of the target not found in this set are displayed in readonly state and they cannot be deleted. The target is allowed to be classified in the classes found in the constraint set of the operation. The user can press a pushbutton, named in the example of figure 8, located above the **COMMIT** button. In this way (s)he gets the contents of the constraint set and (s)he can select the desired from this list (see figure8). A pattern match searching mechanism can search the contents of the list. Alternatively the user can type the node, which is going to be the class of the target, in the text field found next to the pushbutton. Once again the typed class should exist in the constraint set.

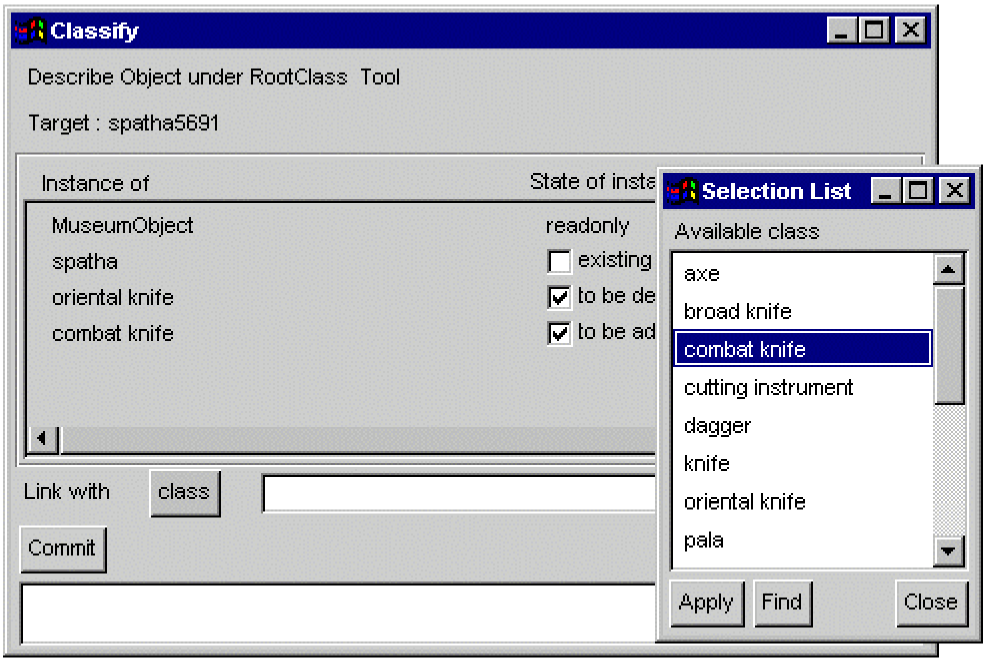

Figure 8: Classify Node

Classify implements an operation for node classification. The target of the operation spatha5691 is classified in the class MuseumObject from which it cannot be deleted (*read-only* state). There is also a link to the class spatha (<*existing*> State). The user is going to delete the classification link to the class oriental knife (*to be deleted* state). (S)he is also going to add a classification link to the class combat knife (*to be added* state). A list of the nodes that can be classes of the target appears by pressing the <*class*> button. The actual update of the information base is done when pressing the **COMMIT** button.

- If there is no constraint set defined, all the links to the classes of the target can be deleted. Also, there is no restriction to the classes that may be added to the target. Every class can be assigned, provided that the semantics of the telos language allows this update (see [DKT95]). In this case the system cannot propose a list of possible classes; no button is offered for this reason. The user only has the ability to type the desired class in the text field.

All changes (links in state <*to be added*>, <*to be deleted*>) are saved in the information base when the user presses the **COMMIT** button.

5.5 Generalize Node

Generalization links from the target node to the superclasses that the operation allows can be created or deleted using the form for node generalization (see figure 9). The same mechanisms with node classification form are offered (see section 5.4).

5.6 Update Node Attributes

Attributes assigned to the target node can be updated using the update attributes form. This form can be alternatively opened by double-clicking on the desired node which appears in the list of the node and operation form, in case an operation of type attribute assignment is define for the current task. Attributes connect the target with nodes or primitive values (integer, real, string, time expression). The node or the primitive value that the attribute points to be called *value* of the attribute, although in case of node the term *to-object* is used too. Attributes appear, as links, which can be added or deleted just, like the classification and generalization links. The difference is that the attributes can have names, they can be classified and generalized in attribute classes and attribute superclasses respectively, they can have attributes of their own, and finally their value (to-object) can be changed. Thus, although the functionality of this form is very much like the functionality of the forms for classification and generalization of a node, there are also some additional features.

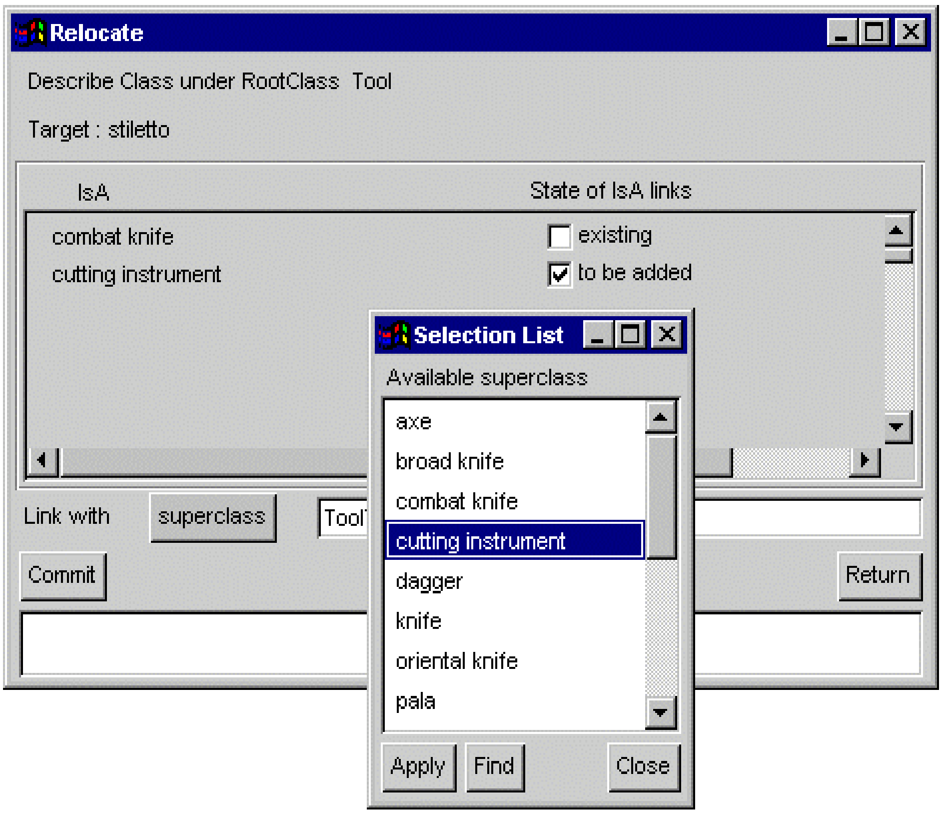

Figure 9: Generalize Node

Relocate implements an operation for node generalization. The target of the operation (*stiletto*) is generalized in the class*combat knife* (<*existing*> State). The user is going to add superclass *cutting instrument* (*to be added* state). A list of the nodes that can be superclasses of the target appears by pressing the smallsuperclass button. Pressing the **COMMIT** button does the actual update of the information base.

The target of the form is indicated by the label **Target** (see figure 10)

The form displays the attributes of the target, which belong to the *updatable attribute classes* [4]. Each updatable attribute class appears as the header of a list (for example the attribute class *method (MeasureType)* in figure 10). The list contains the attributes that belong to this attribute class. The user can see the attributes and their values,

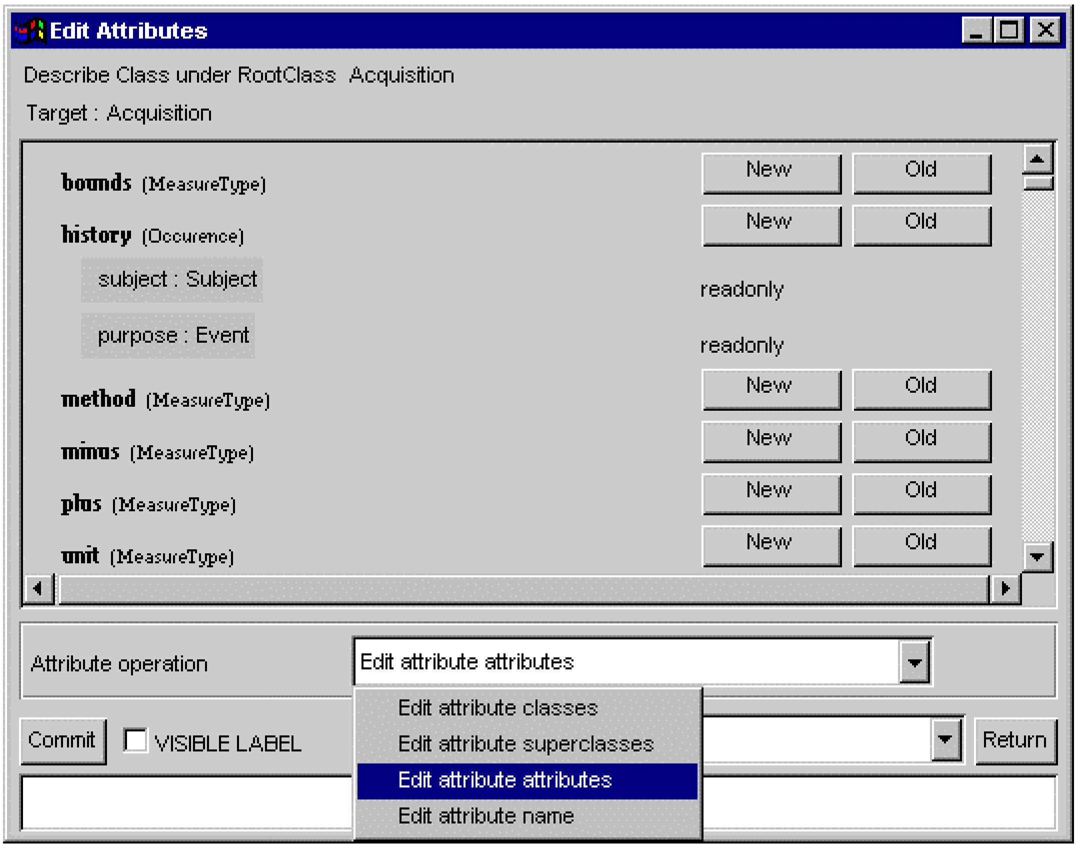

Figure 10: Update Node Attributes

EditAttributes implements an operation for attribute update. The attributes of the target node (*Acquisition*) are updated. The attribute *area_method* pointing to *AreaMethod*is going to be deleted, while the attribute *measure* pointing to *SingleMeasure*is going to be added. Also the user can perform other operations on attributes, such as these that are indicated by the attribute's *operation menu*: **Edit attribute classes, Edit attribute superclasses, Edit attribute attributes, Edit attribute name and Edit attribute to value**

separated by a colon (e.g. for the attribute class *method (MeasureType)*, there are the attributes *time_method*, *volume_method*, *area_method*with values *TimeMethod*, *VolumeMethod*, *AreaMethod* respectively). The object to which the attribute class is assigned appears in brackets, next to the attribute class name (for example the attribute class *method* is assigned to the object *MeasureType*). Buttons for the creation of the attributes is displayed at the right part of the header (the buttons **NEW**, **OLD** for the attribute class *method (MeasureType)* - see also section 5.6.1). The user can create attributes [5] for the updatable attribute classes. (S)he can also perform all the

[4] attribute classes whose instances are allowed to be updated

[5] In the current version, attribute creation cannot be performed for the telos system class *Telos_Object*

available operations on the attributes appearing in the form; (s)he can delete, rename, classify, generalize them, assign attributes to them or change their values (to-objects)

Form states The form has two states, which tune the displayed information: the *attribute name visibility* state and the *attribute class selection* state. The user sets these with buttons found next to the **COMMIT** button.

- *Attribute name visibility* state: A toggle button, which is found next to the **COMMIT** button, allows hiding or showing the labels (names) of the attributes (it is the **VISIBLE LABEL** button in the example shown). It can have two values:
    - **VISIBLE LABEL** denotes visible names of attributes (as shown in figure 10). The user can see the attributes and their values, separated by a colon. It is useful whenever the names of the attributes are important (e.g. at schema level). This state allows all the available operations on attributes.
    - **HIDDEN LABEL** denotes hidden names of attributes. The user can only see the attribute values; not the attribute names. This is useful whenever the names of the attributes are of no importance (usually at data level). This state does not allow operations on attributes other than creation and deletion.

The default value is **HIDDEN LABEL** whenever the target is at data level, and **VISIBLE LABEL** whenever the target is at schema level.

- *Attribute class selection* state: This state is selectable whenever the updateable attribute classes are not (or cannot be) defined in the constraint set of the operation. A cascade button (**INHERITED INCLUDED** in figure 10), which is located next to the *attribute name visibility* state button, allows three possible values:
    - **INHERITED NOT INCLUDED** denotes that the updatable attribute classes are the ones that come from the direct classes of the target.
    - **INHERITED INCLUDED** denotes that the updatable attribute classes are the ones that come from the direct and the indirect classes of the target.
    - **SYSTEM INCLUDED** denotes that the updatable attribute classes are the ones that come from the direct, the indirect and the system classes of the target. The default value is **INHERITED INCLUDED** whenever there are updateable attribute classes coming from the direct and the indirect classes of the target; otherwise the default value is **SYSTEM INCLUDED**. We have to note that in order to retrieve the updateable attribute classes, EF reads the information base. This means that the changes that are not saved in the information base (attributes in state <*to be added*>, <*to be deleted*>) will be discarded for every state transition.

Attribute operation menu There exists a menu, through which the user can select operations for the attributes, other than creation and deletion. It is a pop-up menu and it is visible and selectable only when the *attribute name visibility* state of the form is **VISIBLE LABEL**. It is the menu displayed in figure 10 below the target's list of attributes. The menu has five choices, namely **Edit attribute classes**, **Edit attribute superclasses**, **Edit attribute attributes**, **Edit attribute name, Edit attribute to value**. There exists a current choice in the menu (the last one selected by the user - default namely **Edit attribute classes**) It denotes the operation that is going to be performed whenever an attribute is selected by the user. The attribute selection is achieved by clicking the *right* mouse button on it. The attribute state should be <*existing*>. Then a form corresponding to the current attribute operation appears, as shown in sections 5.6.3, 5.6.4, 5.6.5 and 5.6.6.

5.6.1 Create Attribute

Using the update attributes form, attributes can be assigned to the target. First the class of the attribute has to be chosen. Then, the user has to press the button(s) located at the right of the attribute class header, in order to define the attribute that is going to be created.

The attribute is defined by its value (the node or the primitive value to which it points) and, optionally, its name define the attribute. If the *attribute name visibility* state is **VISIBLE LABEL**, then the user can type the name of the attribute in a text field provided for this reason. As far as it concerns the attribute's value, EF provides choices, depending on the attribute class itself, the attribute class to-object and the settings done in the constraint model (see [Das96a]).

There are nine different buttons, which help the user to specify the value of the attribute that is going to be created. Three buttons deal with nodes **(NEW, OLD, NEW/EXISTING**), five buttons deal with primitive values (**INTEGER, STRING, REAL, TIME, CURRENT**), and while a button named **EDIT COMMENT** deals with free text.

- The **NEW** button denotes that the attribute is going to be created as well as the node this attribute points to (the attribute's to-object). This means that not only an attribute but also a node is going to be created in the information base. First, the class of the node is selected from the -possibly existing- subclasses of the attribute class to-object. By pressing the **NEW** button, the subclasses of the attribute class to-object (except them which are defined in the information base as *System Controlled* classes (see [DT95])) are displayed in an alphabetically sorted scrolling list, and the user can select a subclass from this list. The selected subclass is going to be the class of the attribute's to-object (see figure 11). By clicking the *right* mouse button on **NEW**, the subclass selection is omitted; the attribute class to-object is going to be the class of the attribute's to-object, no matter if there are subclasses for the attribute class to-object. The attribute class to-object is the default class for the attribute's to-object. In some cases, the default class for the attribute's to-object can be set to a class other than the attribute class to-object. In this case the default class has been preselected in the EF constraint model. Once again, by clicking the *right* mouse button on **NEW**, the preselected class appears by default. If more than one default classes are defined in the constraint model, then a scrolling list with these classes appears and the user can select one of them. If there are no subclasses for the attribute class to-object, then the *left* and the *right* mouse button on **NEW** has the same effect, as long as the attribute's to-object class selection is ommited. After the class selection (which may be ommited as already explained) the user has to specify the attribute's to-object (see figure 11). This node is going to be created in the information base, so its name should be unique. The **NEW** button appears alone or together with the **OLD** button.

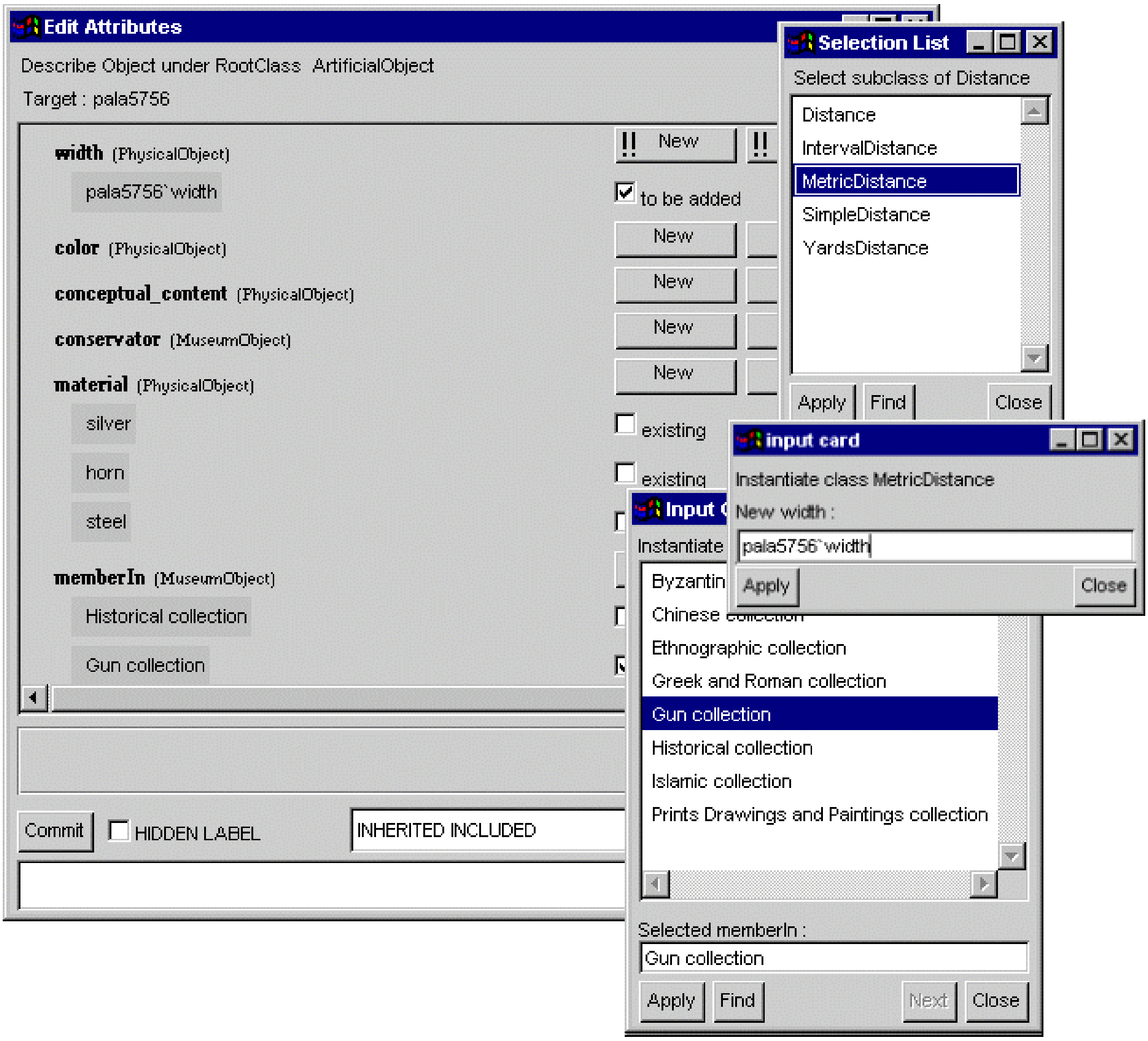

Figure 10: Create Attributes

EditAttributes is an instance of update attributes form. By using this form, the user can create attributes as well as the nodes these attributes point to. See the creation of an attribute for the attribute class *width (PhysicalObject)*. The user presses the **NEW** button in the *width (PhysicalObject)* header. The subclasses of the attribute class to-object appear in a list (see the **Selection List** card). The user selects the subclass *MetricDistance*, which will become the class of the attribute to-object. After pressing the **APPLY** button, an **Input Card** appears. The user types the new node (*pala5756'width*in the example shown) and presses the **APPLY** button. The unnamed attribute pointing from *pala5756* to *pala5756'width under* attribute class *width (PhysicalObject)* is going to be created as well as the node this attribute points to.

The user can also create attributes pointing to already available nodes. See the creation of an attribute for the *memberIn (MuseumObject)* attribute class. The user clicks the *right* mouse button on the **OLD** button in the *memberIn (MuseumObject)* header. The attribute class to-object is the *MuseumCollection*. There is no need to select the subclasses of the attribute class to-object in this case. The user selects the *Gun collection* from the list of *MuseumCollection*'s instances and presses the **APPLY** button. The unnamed attribute pointing from *pala5756* to *Gun collection* under attribute class *memberIn (MuseumObject)* is going to be created.

- The **OLD** button denotes that the attribute, which is going to be created, will point to an already existing node. After the class selection (which is done as in the **NEW** case, but with the *middle* mouse button here, or with double click for 2-

button mouses), the node that the attribute will point to is selected from the set of nodes, which belong to the selected class. The selection is done, once again, from an alphabetically sorted scrolling list (see figure 11) which is opened when the user clicks the *right* mouse button on the **OLD** button. In case of a a *DAG* (Directed Acyclic Graph, see [DT95]) attribute class, the system proposes only the object set of current task for the selection of the node that the attribute will point to. The user can insert (by typing or copy-pasting) the node that the attribute will point to directly, in the text dialog which is opened by clicking the *left* mouse button on the **OLD** button. The **OLD** button appears alone or together with the **NEW** button. The system allows the new attribute to be added, after checking the case of a *DAG* (Directed Acyclic Graph, see [DT95]) attribute class which means that no cycles of this kind of attributes are allowed.

- The **NEW/EXISTING** button denotes that the attribute which is going to be created, will point to a node (existing one or not) of any type, so the system cannot produce a list of available nodes. The user has to type a node as the value of the attribute. If the typed node does not exist in the information base, then it is going to be created together with the corresponding attribute. In this case the user also has to type the class this node will be member of.
- The **INTEGER** button denotes that the value of the attribute must be an integer. This has to be supplied by the user.
- The **STRING** button denotes that the value of the attribute must be a string. This has to be supplied by the user.
- The **REAL** button denotes that the value of the attribute must be a real. This has to be supplied by the user.
- The **TIME** and the **CURRENT** buttons denote that the value of the attribute must be a time expression. By pressing the **TIME** button, the user has to be supplying the time expression. By pressing the **CURRENT** button the system provides the current time. The buttons appear together.
- The **EDIT COMMENT** button denotes that the attribute should point to free text. By pressing this button, a text editor appears and the user can write a text. After saving the text, an attribute with value this text is displayed in *to be added* state.

Whenever an attribute is going to be assigned to the target, it is displayed in *to be added* state. The user can cancel the addition of a *to be added* attribute by pressing the toggle button <*to be added*>, switching it to <*cancel addition*>, just like in the classification and generalization forms.

There may exist some attribute classes, for which it is necessary to have at least one attribute. These are called *necessary* attribute classes. The system informs the user, whenever the latter neglects to set a necessary attribute while updating an object. The necessary attribute classes are displayed at the top of the list of the updatable attribute classes, below the *attribute-like classes* (see section 6.2). The necessary attribute classes may also be distinguished from the others by the "**!!**" image which appears in the button(s) which help the user to specify the attribute that is going to be created.

Furthermore, there may exist some attribute classes, for which it is not allowed to create directed cycles. These are called *Directed Acyclic Graph (DAG) (*see [DT95]) attribute classes. Whenever the addition of an attribute of this kind creates a directed cycle, the user is informed with a warning message and the addition is aborted.

5.6.2 Delete Attribute

Using the update attributes form the user can delete attributes from the target. The attributes that can be deleted are displayed in <*existing*> state. The user demands the deletion of an attribute by pressing the toggle button <*existing*>, switching it to <to be deleted> (see figure 10).

In some cases, the deletion of an attribute may cause the deletion of the value of the attribute. In case of primitive attribute values (integer, string, real, time expression) the deletion of the value is done automatically. The deletion of the node which is the value of a <to be deleted> attribute is allowed only in the following two conditions:

1. the attribute class of the attribute belongs to the attribute metaclass *non_shared*(Individual) (defined in the constraint model). This means that the attribute's value must not be shared with another attribute, and
2. the attribute's value is a node with no attributes, instances, or subclasses of its own and it is not the value of any other attribute in the information base.

There may exist attributes that are not allowed to be deleted; these are displayed in *read-only* state and they are attributes, which have been assigned to superclasses of the target. They can only be deleted through the forms having those classes as targets.

Furthermore, there may exist some attribute classes, for which their deletion causes the automatic deletion of their to-values by the system. These are called *Garbage Collected* (see [DT95]) attribute classes.

Finally, the system demands the existence of at least one attribute for the *necessary* attribute classes (see [DT95]). This means that if the user tries to delete all the attributes of a *necessary* attribute class, EF will prevent the update. In this case, a message dialog box appears, displaying a proper message to the user.

5.6.3 Rename Attribute

Using the update attributes form, the user can rename attributes: after setting the *attribute's operation menu* in the **Edit attribute name** state, the user has to click the *right* mouse button on the attribute that (s)he wants to rename. Then the form for attribute renaming appears (see figure 12)

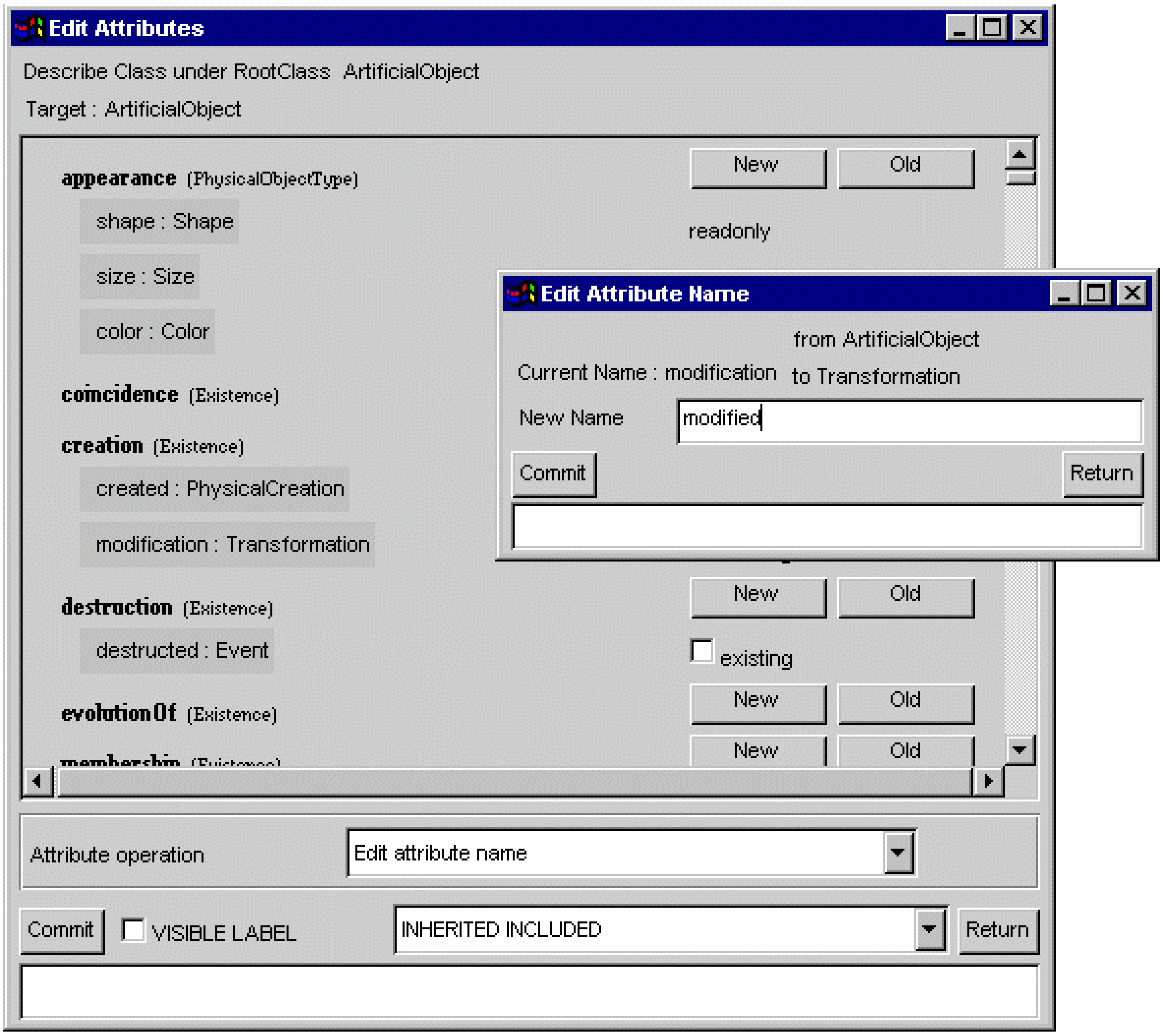

Figure 11: Rename Attribute

The attribute *modification* (pointing from *ArtificialObject*to *Modification*) is the target of the form for attribute renaming. It is indicated by the CurrentName label. It is renamed to *modified* (NewName), upon pressing the **COMMIT** button.

The user has to type the new logical name for the selected attribute. This should be unique in the scope of the from-object. This means that the user cannot re-use a name given to an attribute, which is already, assigned to the target of the update attributes form.

NOTE: In the current version the user cannot rename an unnamed attribute, i.e. set a name for it.

Figure 1 Three-activity process model

5.6.4 Classify Attribute

Using the update attributes form, the user can create or delete classification links from attributes: after setting the *attribute's operation menu* in the **Edit attribute classes** state, the user has to click the *right* mouse button on the attribute (s)he wants to classify. Then the form for attribute classification appears (see figure 13)

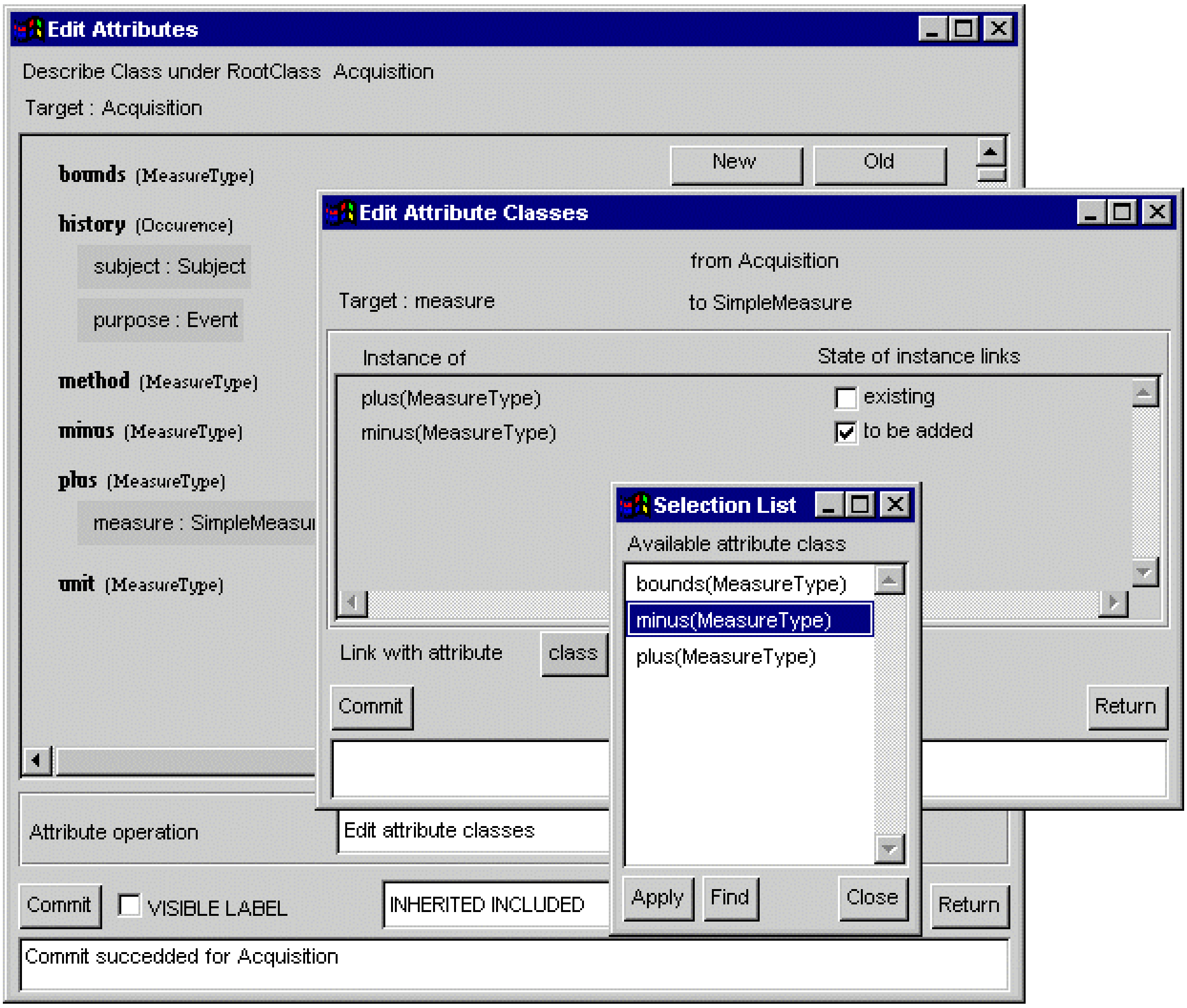

Figure 13 Classify Attribute

The target of the operation for attribute classification (*measure* from *Acquisition* to *SingleMeasure*) is already classified in the attribute class *plus(MeasureType)* (*<existing>* state). A classification link to the attribute class *minus (MeasureType)* is going to be added. The list of the attributes that can be classes of the target appears whenever the user presses the *category* button. The actual update of the information base is done upon pressing the **COMMIT** button.

The functionality of this form is the same as the one of the node classification form (section 5.4). Classification links can be created or deleted from the target. Since the attribute class is difficult to be typed (the user would have to type the attribute class name and the attribute class from-object), there is no text field offered for typing; the user can only select an attribute class from the available list [6], just as (s)he selects classes from the available list in case of node classification.

5.6.5 Generalize Attribute

Using the update attributes form, the user can create or delete generalization links from attributes: after setting the *attribute's operation menu* in the **Edit attribute superclasses** state, the user has to click the *right* mouse button on the attribute (s)he wants to generalize. Then the form for attribute generalization appears (see figure 14)

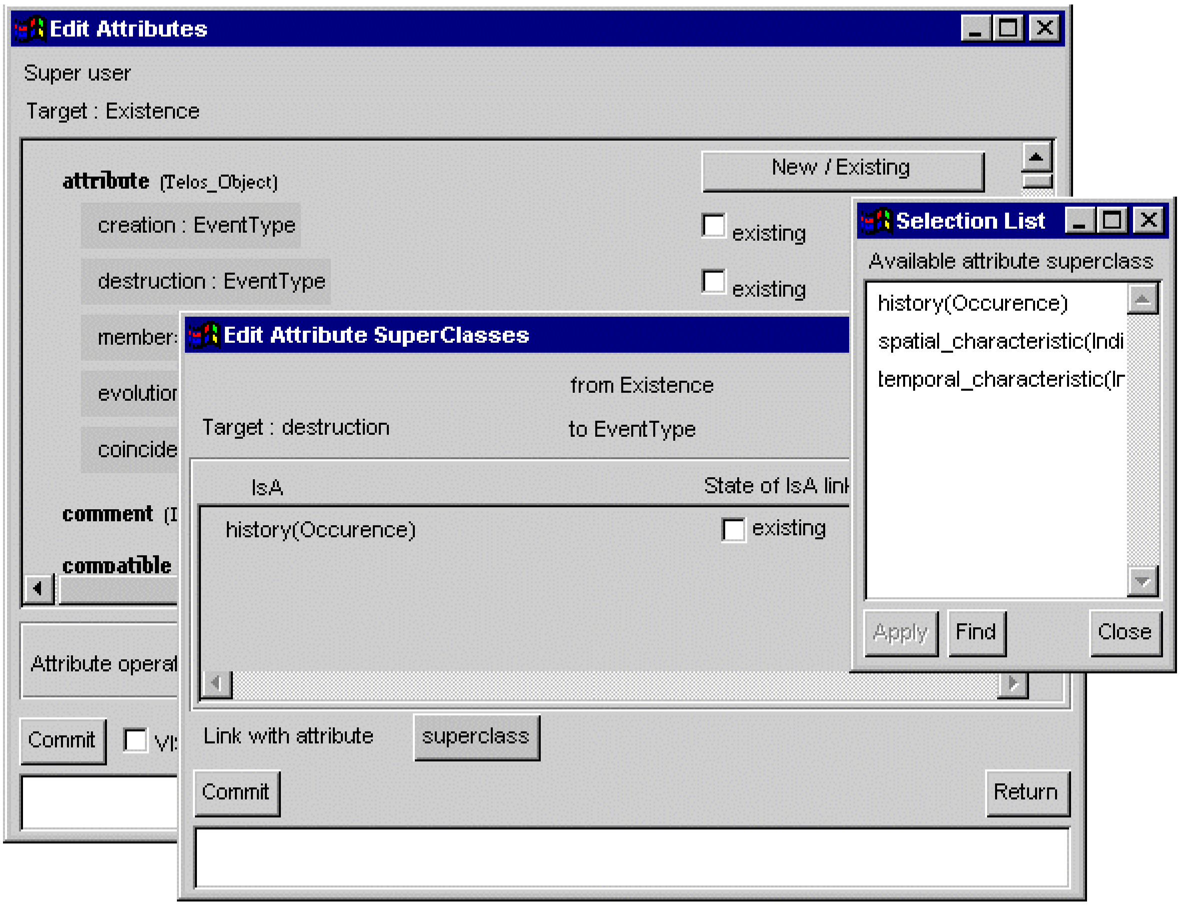

Figure 14: Generalize Attribute

The target of the operation for attribute generalization (*destruction* from *Existence* to *EventType*) already has attribute superclass *history (Occurence)* (<*existing*> state). The list of the attributes that can be superclasses of the target appears when the user presses the **superclass** button.

The same mechanisms with attribute classification form are offered (section 5.6.4)

---

[6] In the current version, no attribute class whose TO is a telos system class, is offered for selection.

5.6.6 Update Attribute Attributes

Using the update attributes form, the user can update attribute attributes: after setting the *attribute's operation menu* in the **Edit attribute superclasses** state, the user has to click the *right* mouse button on the attribute whose attributes (s)he wants to update. Then the form for updating the attribute attributes appears (see figure 15)

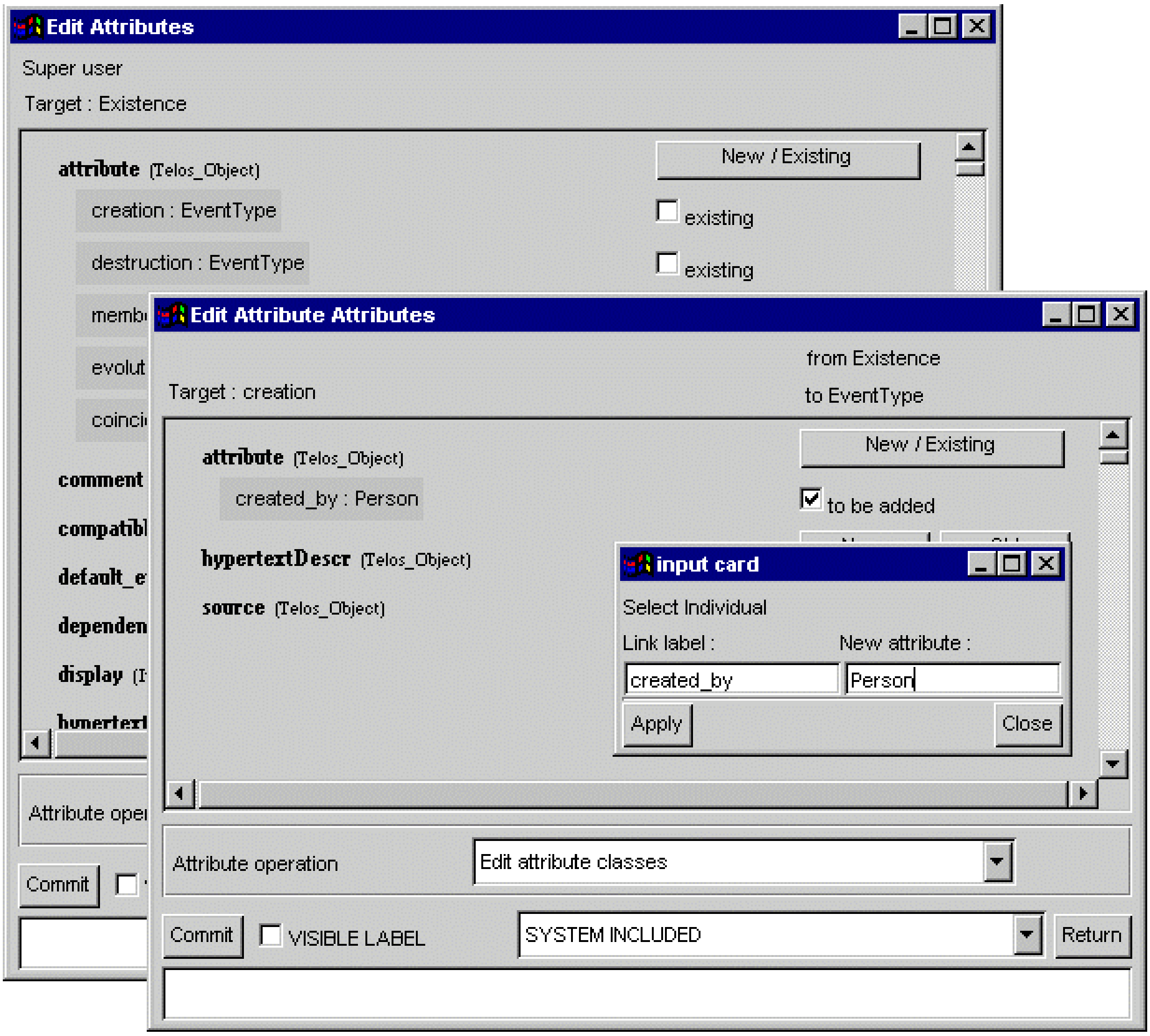

Figure 15: Update Attribute Attributes

The attributes of the target attribute (*creation* from *Existence* to *EventType*) are updated. The attribute *created_by* pointing to *Person*is going to be added. Also the user can perform other operations on the attributes of the target. These operations are indicated in the *attribute's operation menu*, just like in updating the attributes of a node.

The user can update the attributes of the target attribute, just as (s)he can do in case of updating the attributes of a node (section 5.6) In the form for updating the attributes of an attribute, the updatable attribute classes are not defined in the process model; these are selected by the user, using the *attribute class selection* menu.

5.6.7 Update Attribute Value (to-obiect)

Using the update attributes form, the user can update an attribute's value (to-object): after setting the *attribute's operation menu* in the **Edit attribute to value** state, the user has to click the *right* mouse button on the attribute (s)he wants to update. Then the form for attribute's to-value renaming appears.

The same mechanisms with attribute renaming form are offered (section 5.6.3)

5.7 User Defined Operations

The user can construct special update forms for special operations related to his (her) application (see [DT95]).

6. Further features

EF further provides the following mechanisms, which facilitate the update of the information base: complex object update (section 6.1), attribute-like classes update (section 6.2), and primitive value editing (section 6.3) and free text editing (section 6.4).

6.1 Complex object update

Some objects linked with attributes, may be considered as a complex object. The objects that are parts of a complex object are called dependent objects. The attributes through which these objects are connected, are called dependent attributes. The objects that have such a property are specified in the constraint model, as explained in [Das96a]. The user can distinguish the dependent attributes from the others by the "▶" image which help the user to specify the attribute that is going to be created. Additionally, these attributes are displayed at the top of the attribute list, below the *attribute-like classes* (see section 6.2) and the *necessary attributes* (see section 5.6.1). EF provides mechanisms, in order to update the dependent objects easily. There can also be an attribute, which is dependent and necessary. The user can distinguish the dependent - necessary attribute from the others by the "**!**▶" image which help the user to specify the attribute that is going to be created.

6.1.1 Creation of dependent objects

Dependent objects can be updated through the *dependent object update attributes form* (see figure 16)

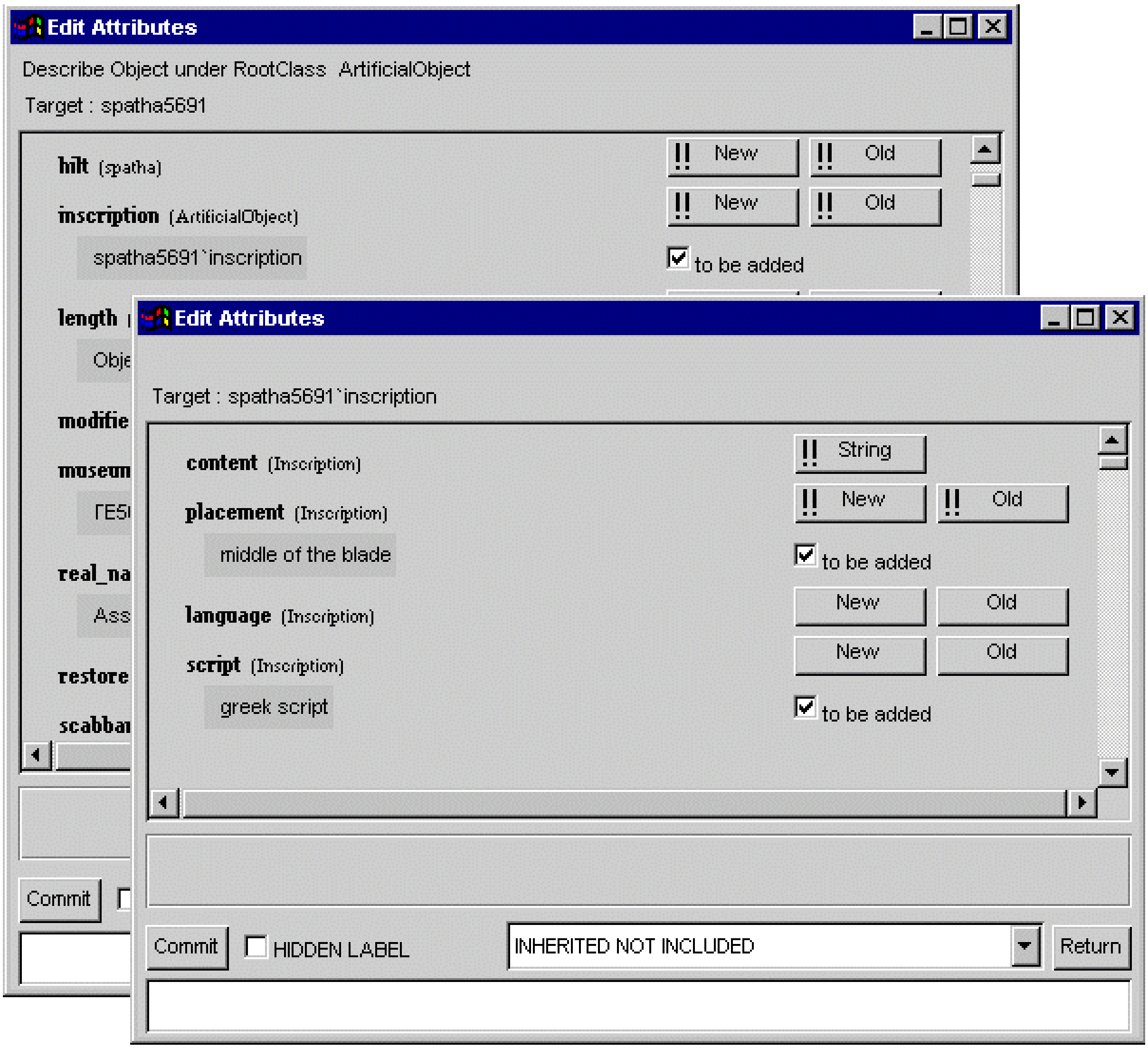

Figure 16: Create dependent object

The object spatha5691'inscription depends on the object spatha5691. The *dependent object update attributes form* appears for the former, when the user clicks the *right* mouse button on it. Then a chain of update attributes forms appears. The user can edit the attributes of the dependent object. By pressing the **COMMIT** button in the last form of the chain, the information base is updated with the changes of the chain's forms in one transaction.

This form can be called from every update attributes form. The user has to click the *left* mouse button on the value of an attribute [7]. The attribute state should be <*existing*> or <*to be added*>.

When a new dependent object is going to be created in the information base, then the *dependent object update attributes form* is displayed automatically. In this case, the name of the dependent object is proposed by the system.

Creating dependent objects, a chain of update attributes forms appears. By pressing the **COMMIT** button in the last form of the chain, the information base is updated with the changes of the chain's forms in one transaction.

6.1.2 Deletion of dependent objects

The user can delete the dependent object from the *dependent object update attributes form*. The deletion is allowed, provided that there are no attributes assigned to the dependent object, and that there are no more than one attribute pointing to it (there should exist at least the attribute that connects the dependent object with the object on which it depends). By pressing the **DELETE** button located at the right top of the form, deletion of the dependent object takes place (see figure 17).

After the deletion of the object, the *dependent object update attributes form* closes.

6.2 Attribute-like classes update

Quite often the user classifies a node in order to assign properties to it; then classification in a sense plays the role of attribution. The classes that are used for property assignment can be displayed in the update attributes form. These classes are called *attribute-like classes* and they can be preselected in the EF's constraint model (see [Das96a]). In this case, classification links can be grouped in categories and can be displayed like attributes in separate lists, just like the attribute class lists shown in section 5.6. The user has the ability to select available attribute-like classes, which can be added to the target just like the attributes, by pressing the **OLD** button. Attribute-like classes are displayed at the top of the list of the updateable attribute classes.

6.3 Primitive value editing

EF provides for editing the primitive values of the attributes. When the value of an attribute is primitive (integer, string, real, time expression, indicated by the buttons **INTEGER, STRING, REAL, TIME** respectively), the user can edit it by clicking the *left* mouse button on it. Then, if the attribute state is <*existing*> or <*change value*>, a text field appears. It contains the current primitive value. The user can edit this value and apply the change back to the form by pressing the **APPLY** button. The attribute is then displayed in state <*change value* > (see figure 18).

The user can cancel the change of the value, by pressing the toggle button <change value>, switching it to <cancel value change>.

---

[7] We remind that by clicking the *right* mouse button on an attribute, the user can update it with the operation being current in the *attribute's operation menu*.

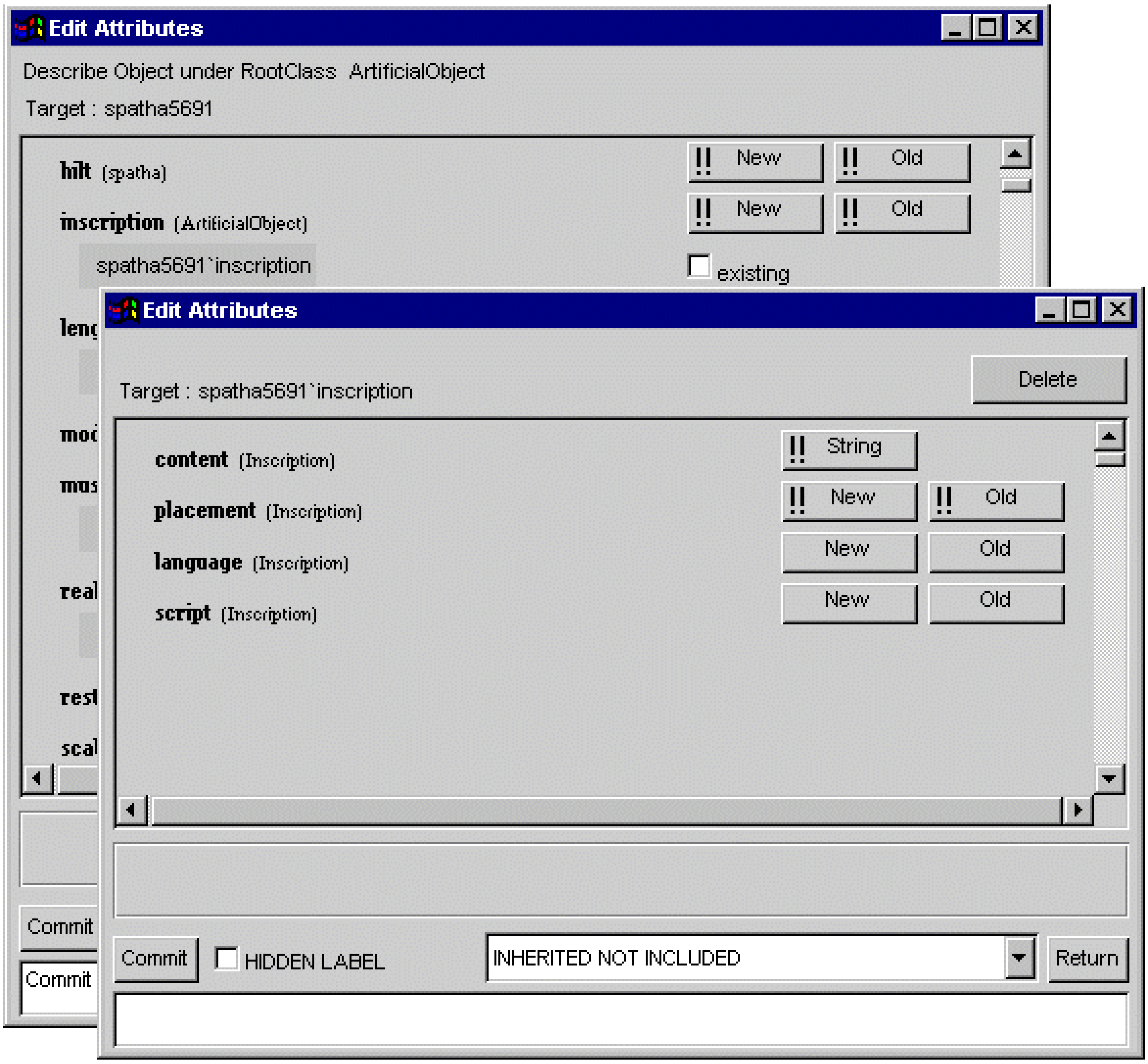

Figure 17: Delete dependent object

The dependent object *spatha5691'inscription* can be deleted from the *dependent object update attributes form* by pressing the **DELETE** button.

6.4 Free text editing

EF provides for editing the free text assigned to a comment type attribute (indicated by the **EDIT COMMENT** button). The user can click the *left* mouse button on the attribute. Then, if the attribute state is <*existing*> or <*to be added*>, a text editor appears and the user can edit the current text value of the attribute. After saving the text, the attribute is displayed in <*to be added*> state. The user can cancel the editing, by pressing the toggle button <*to be added*>, switching it to <*cancel addition*>.

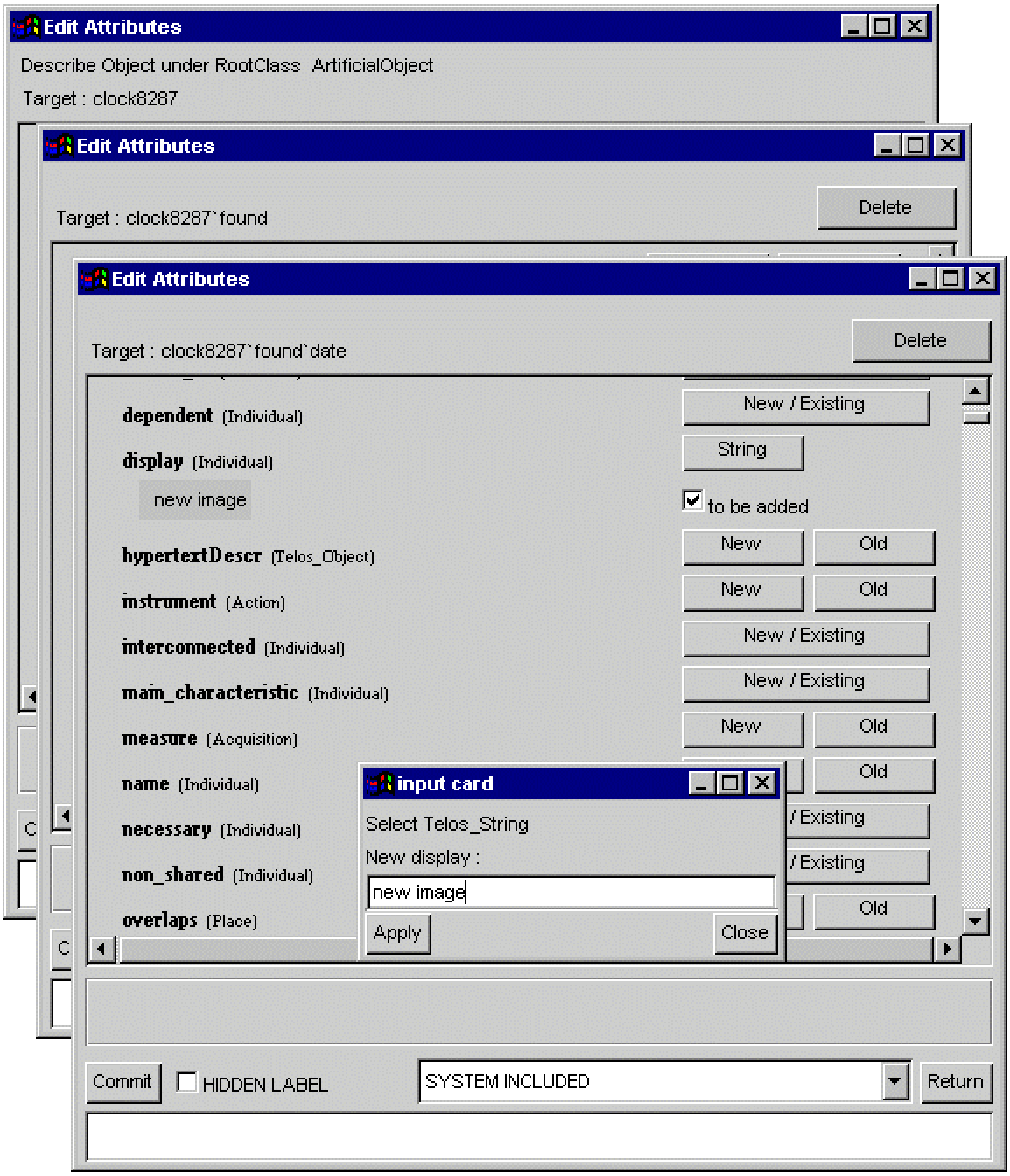

Figure 18: Primitive value editing

The user can edit the primitive value, using the editor offered for this reason. It appears when the user clicks the *left* mouse button on an attribute pointing to a primitive value (the time expression *<1970 June 5>* in the example shown). After editing, the user can press the **APPLY** button, and the attribute's state is changed to <change value> (as shown in the figure). The primitive value is changed by pressing the **COMMIT** button.

7. References

[DKP95] Dimitris Daskalakis, Polykarpos Karamaoynas, and Nikos Prekas. *"SIS Graphical Analysis Interface User's Manual"*. Information Systems and Software Technology Group, Institute of Computer Science Foundation of Research and Technology Heraklion, Crete, Hellas, November 1995, version 1.3.

[DKT95] Martin Dörr, Polivios Klimathianakis, and Manos Theodorakis. *"SIS Data Entry Language User's Manual"*. Information Systems and Software Technology Group, Institute of Computer Science Foundation of Research and Technology Heraklion, Crete, Hellas, November 1995, version 1.3.

[DT95] Dimitris Daskalakis and Yannis Tzitzikas. *"Customizing Data Entry Forms"*. Information Systems and Software Technology Group, Institute of Computer Science Foundation of Research and Technology Heraklion, Crete, Hellas, November 1995, version 1.3.

[MBJK90] John Mylopoulos, Alex Borgida, Matthias Jarke, and Manolis Koubarakis. *"Telos: Representing Knowledge about Information Systems"*. ACM Transactions on Information Systems, 8(4), October 1990.

[The95] Maria Theodoridou. *"Binding SIS with External Tools"*. Information Systems and Software Technology Group, Institute of Computer Science Foundation of Research and Technology Heraklion, Crete, Hellas, November 1995, version 1.3.

8. INDEX

R

S

T

U

V

9. Appendix A -Changes from previous versions

Changes from version 1.3 to version 2.2

- User defined operations specification (see [DT95])
- System Controlled classes (see [DT95])
- Forwards/Backwards Sorted attribute classes (see [DT95])
- Prefix mechanism (see [DT95])
- Directed Acyclic Graph (DAG) attribute classes (see [DT95])