A simple yet useful approach to implementing UML Profiles in CASE tools (Extended Version)

Jordi Cabot, Cristina Gómez

Universitat Politècnica Catalunya
Departament de Llenguatges i Sistemes Informàtics
Jordi Girona 1-3 E08034 Barcelona (Catalonia)
{jcabot|cristina}@lsi.upc.es

Abstract

UML Profiles allow designers to customize the UML to their particular domain or purpose. This is a key feature to adapt the models of the information systems to the continuously evolving organizations that they represent. These profiles must be implemented in CASE tools the designer employs to gain the benefits of its use.

In this paper we propose a design strategy to overcome the limitations of current CASE tools respect to profiles definition. The basic idea of our method is to extend the CASE tool with a set of profile operations that act as an interface with the designer in order to allow him to define and evolve models that use elements of the profile.

Keywords

UML Profiles, CASE tools

Table of Contents

1. Introduction

The extension mechanisms provided by some conceptual modeling languages [Conc03,OMG01b,Telo90] allow the designers to customize them to its particular domain or purpose. This is a key feature to adapt the models of the information systems to the continuously evolving organizations that they represent. In the UML, a coherent set of such extensions constitutes an UML Profile [OMG01b].

In recent years, many profiles have been developed to support different tasks of software systems development. More recently, the concept of profile has been paid much attention due to its importance in the context of the Model Driven Architecture [OMG01a]. MDA is aimed at generating automatically (or semi-automatically) design and implementation models of a system in multiple platforms. This is done by applying mappings to the specification system models. These mappings are expressed using UML Profiles.

Some profiles (for instance, [OMG99,OMG02b,OMG02c]) have been already standardized (or are in process) by the OMG while others can be considered standards *de facto* [Cona99, NaMa01]. Some CASE tools have incorporated these standard profiles, allowing its users to take profit of them in their models.

However, other profiles, which are not standardized, cannot be used because the majority of tools do not incorporate them. Moreover, tools neither completely support completely new profiles definition. This reduces, dramatically, the benefits that could be gained with the use of profiles. In fact, the proliferation of a huge amount of profiles has not been accompanied by the necessary support of existing CASE tools.

The main goal of this paper is twofold. On the one hand we identify the limitations of CASE tools when a profile implementation is considered. On the other hand, we propose some design strategies to avoid these limitations in order to allow designers to define and evolve models that use elements of the profile.

The rest of this paper is organized as follows. Section 2 reviews the concept of profile and defines the *Presentation* profile that will be used as an example throughout the paper. Section 3 focuses on the limitations of CASE tools and proposes some design strategies to overcome them. These strategies are applied to the *Presentation* profile. Moreover, it presents a comparison with other possible approaches. In Section 4, the *Presentation* profile is implemented using the Objecteering/UML CASE tool. Finally, last section remarks some conclusions and points out future work.

2. UML Profiles

An UML Profile is a package that contains model elements that have been customized for a specific domain or purpose using extension mechanisms, such as stereotypes, tagged definitions and constraints [OMG01b, 2-191].

A stereotype provides a way of defining virtual subclasses of UML metaclasses with additional semantics. It can define additional constraints over its base metamodel class as well as tags to define additional properties.

A constraint is a semantic restriction represented as a text expression. Constraints are attached to one or more model elements. They are usually specified using the OCL (Object Constraint Language) [OMG01b, Chapter 6].

Tags definitions specify new kinds of properties as part of a stereotype definition. The actual properties of individual model elements are specified using tagged values.

Stereotypes and tags definitions can be derived. This means that they can be computed from other elements by means of a derivation rule. The derivation rule can be written in OCL as a constraint [RuJB99].

Figure 1 presents the *Presentation* profile, an UML Profile that will be used as an example throughout the paper. The purpose of this profile is to describe the presentation model of an application. Specifically, it aims at the design of its graphical user interface (GUI). Note that it is not the goal of this paper to define a complete presentation profile. We only use it as a running example. The basic concepts of this profile are mainly taken of [HeKo01, Cona99].
We assume that a GUI consists only of forms and two types of controls: edit boxes and buttons.

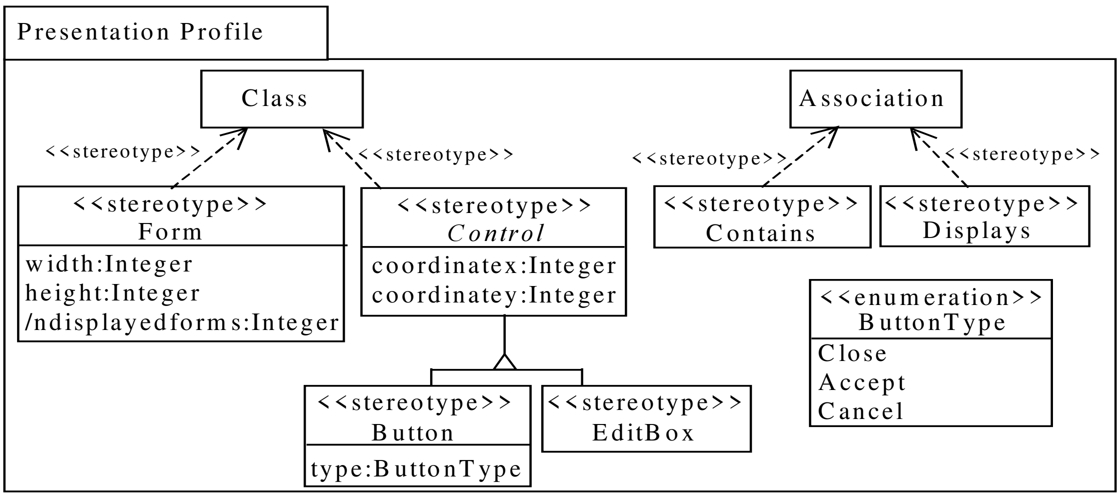

Figure 1: Presentation profile

A form is used to introduce and visualize data of an information system. It may contain edit boxes and buttons and may display other forms. The *Form* stereotype represents the forms that appear in a GUI. The base tags represent the *width* and *height* of the forms. Moreover, this stereotype has a derived tag *ndisplayedforms* that represents the number of forms displayed by the form.

The *Control* stereotype represents the controls that appear in forms. This stereotype is abstract, and it is used only to define two common tags: *coordinatex* and *coordinatey*. These tags represent the coordinates x and y of the control inside the form. The binary relationship of containment between forms and the controls placed inside is specified by means of an association with the *Contains* stereotype.

A single instance of a class with the *EditBox* stereotype represents an edit box control of a form, while a class with the *Button* stereotype represents a button control. Its tag *type* indicates if the button is an accept, close or cancel button.

The relationship between two forms where one of them (source) displays the other (destination) is specified by means of an association with the *Displays* stereotype. We use the navigability in the association to distinguish between the source and destination forms. The association can be traversed from the source to the destination form.

The profile includes several constraints concerning the stereotypes (their formalization in OCL can be found in the appendix A):
R01: The height and width of a form must be positive.
R02: There cannot exist a cycle of associations with the *Displays* stereotype.
R03: The coordinates x and y of the controls cannot be out of the range of the forms where they are placed inside and must be positive.
R04: The associations with the *Contains* stereotype must have two participants. One of them must be a form and the other must be a control.
R05: There cannot be two *Contains* associations between the same form and control.
R06: A *Displays* association must have two participants. Both participants must be forms. The navigability is single in the direction of the destination form.
R07: There cannot be two *displays* associations between the same forms.
R08: The multiplicity of the associations with the *Contains* and *Displays* stereotype must be 1 in both association ends.

As an example, Figure 2 shows a simple GUI, specified using the *Presentation* profile, for an application that deletes customers from the information base. This application consists of two forms: *SelectCustomer*, to select the customer to be deleted, and *DeleteCustomer*, to delete the chosen customer. In both forms, the user can accept or cancel the action of selecting (deleting) the customer.

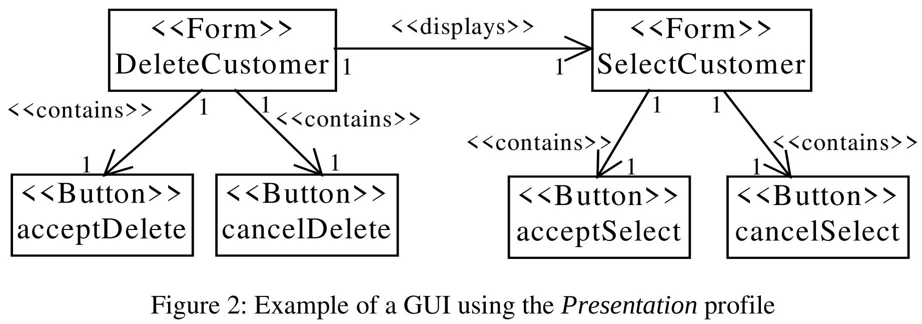

Figure 2: Example of a GUI using the *Presentation* profile

3. Designing an UML Profile

Nowadays, it is difficult to imagine that a company that uses the UML notation in the analysis and design of its applications does not employ also a CASE tool to gain the benefits of its use (for instance, round-trip engineering, documentation generation and so on). Therefore, the profiles, to be useful, must be incorporated into the CASE tool the company uses. In this section, we propose a general strategy to design a profile in order to be easily implemented in CASE tools.

There is a huge amount of different CASE tools (see [ObyD03] or [Jeck03] for an exhaustive list) but most of them share the same features and limitations relating to profiles definition. Hence, in the design stage we must take into account the characteristics of the CASE tools that may affect the implementation of the profile on them.

In the next subsections we will identify such limitations, describe how they affect the implementation of the profile and purpose design strategies to overcome them for each component of a profile: stereotypes, tags, constraints and derived elements.

3.1. Stereotypes and tags

Most tools support the concept of stereotype. So, they can be implemented as such on the CASE tool. In fact, if our tool does not allow the definition of the stereotypes, it makes no sense to try to implement UML Profiles, since they are based on the notion of stereotypes. The same happens with the definition of tags in the stereotypes.

With stereotypes and tags we can already specify graphically models that use the profile. However, we need to define the constraints and derived elements of the profile to ensure that it is correctly used and offers the behavior expected by the user.

3.2. Constraints and derived elements

Most CASE tools are able to verify the constraints defined in the UML metamodel. Thanks to this feature, we can specify well-formed models. For instance, we could not define two attributes with the same name within a class because this violates a well-formedness rule of the *Classifier* metaclass [OMG01b, pp. 2-56]. On the contrary, to the best of our knowledge, none of the present CASE tools is able to enforce the constraints attached to the stereotypes of the profile. Some tools allow its definition (for example [Obje03]) but are unable to interpret them. Few tools like [Argo03] and [USE03] offer an OCL interpreter for checking the syntax of the constraints applied to a model. [USE03] goes even a step forward and allows the designer to create instances of the model to validate the constraints. However, these tools only apply these techniques at the model level but not to the metamodel itself, and thus, we cannot use them to check the constraints attached to the stereotypes of the profile.

Designers specify and evolve its models using the operations provided by the tool (for instance create a class, create an association, assign a stereotype, add an attribute, etc), usually through a graphical interface. If the designer uses a profile using those tool operations and the tool is unable to verify the profile constraints, we cannot ensure that the resulting model is consistent with them. For example, using the tool operations we could specify a model where a form displays an edit box, violating rule R06.

To avoid this problem, we propose to design a set of profile operations that act as an interface with the tool user when s/he wants to use some component of the profile in order to ensure the consistency of its use. These operations will be in charge of enforcing the correct use of the profile. In principle, we must create the following set of operations for each stereotype of the profile:

- An operation to create instances of the stereotype.
- An operation to delete instances of the stereotype.
- For each base tag of the stereotype, an operation to be able to modify its value.

We propose to design these operations by means of contracts. An operation contract describes changes in the state of the system when the operation is invoked. A way to define a contract is defining its pre and postconditions. Preconditions are the conditions that must be satisfied before the operation execution. Postconditions are the conditions that must be satisfied after the operation execution.

Each profile operation will have as preconditions the profile constraints that the operation may violate. So, if the user attempts to execute an operation that would lead to an inconsistent model, the preconditions of the operation would forbid it.

CASE tools neither support the management of derived information, and thus, they are not able to do an automatic computation of the derived tags or stereotypes of the profile. In general, there are two different approaches to deal with derived information as far as the design of the information system is concerned: it can be either computed by means of an operation whenever it is requested or materialized by explicitly storing its contents in the information base. Since we cannot define operations as part of the stereotype definition, materialization is the only possible approach. Therefore, we must define the derived tags and stereotypes as base ones and update their values according to the changes that would be induced by the derivation rules.

As in the previous case, to ensure the values of these derived elements are up to date, we should develop a set of operations that act as an interface with the designer when s/he wants to use some component of the profile. These operations must guarantee the correct calculus of the new values of derived elements. However, we do not need to develop such operations because they are the same used to check the constraints of the profile. We add the calculus of the new values of the derived elements as postconditions of the corresponding operations.

For instance, one of the profile operations for the *Displays* stereotype is *CreateDisplays(sf:Class, df:Class)*. This operation creates an instance of an association with the *Displays* stereotype between the source form *sf* and the destination form *df*. The preconditions of this operation must ensure that profile constraints related to the *Displays* stereotype are satisfied. In this case, the preconditions must ensure that that *sf* and *df* are forms, there not exists already a *Displays* association between them and that the new *Displays* instance does not causes a cycle of *Displays* associations. The postconditions of this operation must guarantee the correct calculus of derived elements that can be affected by the creation of the *Displays* association. In this case, the postconditions guarantee that the source form *ndisplayedforms* tagged value has been increased by one.

After assigning the constraints and derivation rules to each profile operation we may find that some operations do not have any preconditions or postconditions (for example, if the stereotype does not participate in any constraint or derivation rule). In such cases, instead of implementing it we can use the tool operations for the base class of the stereotype.
Figure 3 shows the interaction between the designer and the CASE tool. The designer uses the tool operations to specify its models, unless s/he wants to use some of the elements of the profile. In that case, s/he uses the new profile operations. Note that, the design of the profile operations and its subsequent implementation implies an extension of the CASE tool functionality.

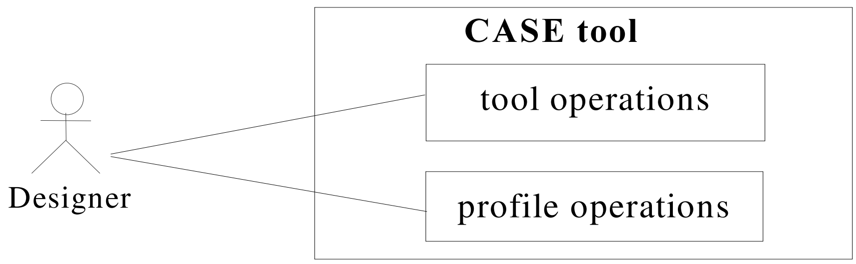

Figure 3: Interaction between the designer and the CASE tool

3.3 Designing the Presentation profile

According to the design process explained above, in the *Presentation* profile it is necessary to materialize the tag *ndisplayedforms* of the *Form* stereotype and define the following operations: *CreateForm, , CreateButton, CreateEditBox, CreateContains, CreateDisplays, ModifyWidth, ModifyHeight, ModifyCoordx, ModifyCoordy* and *RemoveDisplays.*
It does not appear other operations as are: *RemoveEditBox, RemoveButton, RemoveForm, ModifyTypeButton or RemoveContains*. In this profile, they are not necessary since the tool operations are enough. For instance, the deletion of a *Contains* association does not involve any constraint or derivation rule, and thus, the operation for removing an association already provided by the tool is good enough.

As an example, we show here the design of the *CreateDisplays* operation. As we have explained in previous section, this operation must verify, in its preconditions, some profile constraints (R02, R06 and R07) before creating the *Displays* association. Once created, the postconditions calculate the new value of the derived tag *ndisplayedforms* of the source form. In the following, we present the design of the *CreateDisplays* operation contract in OCL:

```
context System:: CreateDisplays(sf:Class, df:Class)
pre R02:
not df.allDisplayedForms->includes(sf)
pre R06:
sf.stereotype->one(name='Form') and df.stereotype->one(name='Form')
pre R07:
Association.allInstances->select(stereotype->one(name='Displays') and
connection.participant->asSet()= Set{sf,df})->isEmpty()
post Create displays association:
dis.oclIsNew()
and dis.oclIsTypeOf(Association)
and dis.stereotype=Stereotype.allInstances->any(name='Displays')
and aesf.oclIsNew()
and aesf.oclIsTypeOf(AssociationEnd)
and aesf.isNavigable=true
and aesf.multiplicity.range->size()=1
and aesf.multiplicity.range.lower=1
and aesf.multiplicity.range.upper=1
and aedf.oclIsNew()
and aedf.oclIsTypeOf(AssociationEnd)
and aedf.isNavigable=false
and aedf.multiplicity. range.lower=1
and aedf.multiplicity.range->size()=1
and aedf.multiplicity. range.upper=1
and dis.connection->first()=aesf
and dis.connection->last()=aedf
and aesf.participant=sf
and aedf.participant=df
-- Induced effects ndisplayedforms:
and sf.taggedValue->select(type.name='ndisplayedforms').dataValue=
sf.displayedForms->size()
```

The complete design of the other profile operations can be found in appendix B.

3.4 Alternative approaches

Instead of preventing an incorrect use of the profile by means of the profile operations, we could let the designer use the profile using the tool operations and, after, verify if the model is consistent with respect to the profile constraints.

We see, at least, two different ways of implementing this delayed consistency check. We could either done it by developing a single profile operation in the tool in charged of checking all the profile constraints, or by exporting the model to a XMI document (XML Metadata Interchange, [OMG02a]) and then, analyzing the resulting XMI file by means of some kind of application able to interpret its content and checking the profile constraints. This latter process is sketched in Figure 4.

XMI is an interchange format for metadata that is defined in terms of the Meta Object Facility (MOF) standard [OMG02, pp. 1-2]. As shown in [Jeck03], there exists already some tools that offer the possibility of importing and exporting a model into an XMI document. The main advantage of this approach is that, since XMI is a standard specification, we would be able to use a single application to check XMI models generated by any CASE tool. This application could take profit of some OCL interpreters already developed. Of a significant importance is [OCLE] an object constraint language evaluator able to import an XMI model and ensure that verifies the constraints previously defined in the tool.

Unfortunately, nowadays, the XMI documents generated by most tools present some differences from the XMI standard and are seldom interchangeable. Therefore, we will be forced to develop a different *checking* application for each tool we want to use.
The major drawback of both alternatives is that the designer may be using an inconsistent model during a period of time, until s/he checks it. Moreover, in general, the possible model errors cannot be corrected automatically, so the designer would be responsible of fixing them (and repeating the process to ensure that s/he has fixed the errors correctly). Notice, also, that the above alternatives are unable to deal with derived elements.

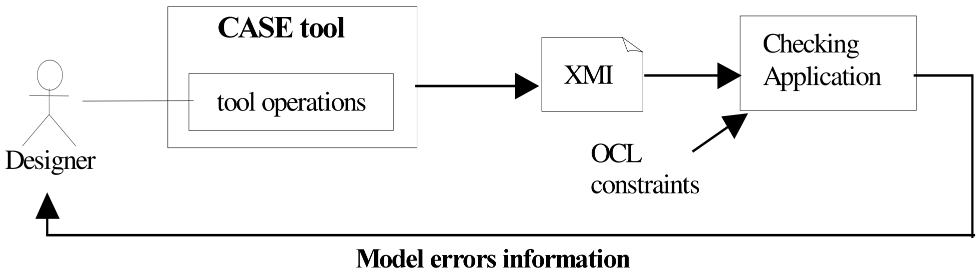

Figure 4: Delayed checking of the consistency of a model using XMI

4. Implementing an UML Profile

Once we have designed the profile, we are ready to implement it in a CASE tool. In the implementation phase, we must adapt the design to the specific characteristics of the target CASE tool.

As we have assumed before, CASE tools allow the definition of stereotypes and tags, and thus, this part of the profile will be easily implemented. Moreover, most of them also offer scripting facilities that allow us to implement the profile operations. With scripting, the designer may create scripts that can access and modify the models within the tool. To do it, the scripts must use some kind of metamodel interface offered by the tool.
Then, the adaptation process basically consist of two steps:

- Implementing the operations specified in OCL to the specific scripting language offered by the tool.
- Modifying the operation code, if the metamodel interface offered by the tool differs from the UML metamodel, which has been used in the operation design.

In this process, we could use any CASE tool that allows scripting. However, when the tool offers a proprietary metamodel interface the process of changing the operations design may require a significant effort. Thus, we should select an UML CASE tool that offers an interface as similar as possible with the UML metamodel. For instance, Objecteering [Obje03] is a tool that offers to its users an interface that follows the UML Metamodel. Thereby, with this tool, we do not need to do the second step (in fact, we will still need to do some minor changes because the Objecteering metamodel presents some small differences respect to the UML metamodel).

In the following, we implement the *Presentation* profile with the Objecteering/UML CASE tool, in particular using the UML Profile Builder (one of the tools included in Objecteering/UML). Once the new profile is implemented, it can be exported to be used by designers through the Objecteering/UML UML Modeler (the Objecteering/UML modeling tool for software developers).

4.1 Implementing the *Presentation* profile in Objecteering/UML

To create a new profile we must define the set of new stereotypes with its tags and the operations that will act as an interface with the user. In this case, there are six stereotypes (*Form, Control, EditBox, Button, Displays* and *Contains*), and ten operations (*CreateForm, CreateButton, CreateEditBox, CreateContains, CreateDisplays, ModifyWidth, ModifyHeight, ModifyCoordx, ModifyCoordy* and *RemoveDisplays*) to implement.

The definition of the stereotypes is done by tool interaction. The only thing to do is to provide all the stereotype information (base class, tagged values, etc).

Once the new stereotypes have been defined, we are ready to implement the operations. These operations are implemented using J methods. The J methods are methods attached to a specific metaclass, and executed using an instance of that metaclass. They are written in the J language. The J language is the proprietary tool scripting language. Nevertheless, its "Java-like" syntax makes it easy to use. It presents some interesting features (see [Soft99] for more details) like model handling capacities. Thanks to these features, the J language is perfectly suitable for implementing the profile operations. In fact, everything the designer can achieve through Objecteering/UML UML Modeler can be achieved through programming with the J language.

The J method corresponding to the *CreateDisplays* operation, previously designed in OCL (see section 3.3), can be found in the appendix B. Note that the method code checks all the constraints and computes the derived information defined in the operation design.

Once the profile and its operations have been implemented, the designers are ready to interact with the CASE tool in order to use them in their models. If the designer wants to create a new *Displays* association, s/he must invoke the *CreateDisplays* operation. Figures 5 and 6 show the process of creating a *Displays* association between two forms previously defined. First (Figure 5), we select the source form and execute the operation. Then (Figure 6), the designer through a dialog box provides the name of the destination form. The result is shown in Figure 7. As a consequence of the operation execution the *ndisplayedforms* tagged value has been increased by one.

The rest of the operations are implemented and used in a similar way. Their complete code can be found in appendix B.

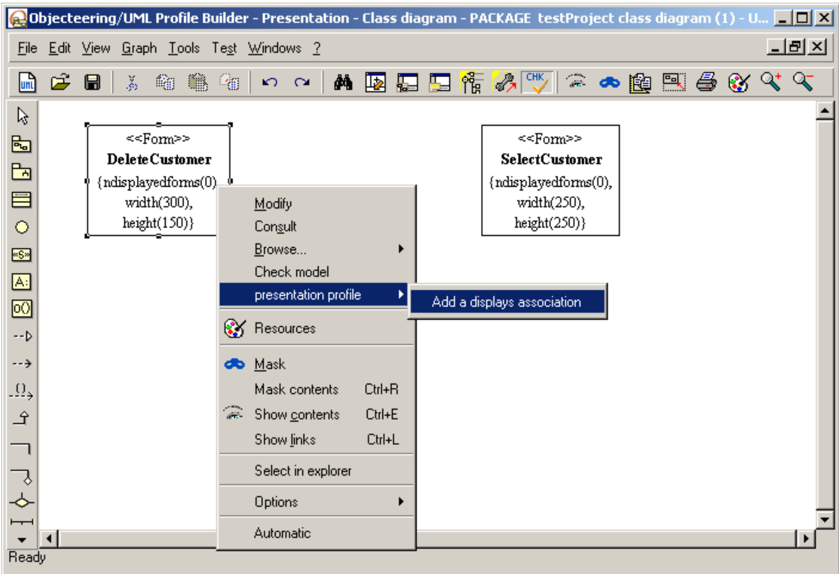

Figure 5: Example of creating a displays association

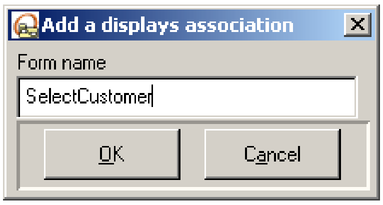

Figure 6: Selecting a destination form

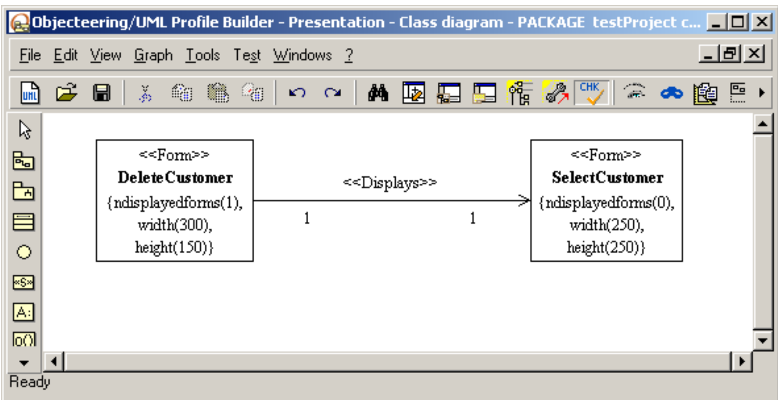

Figure 7: Result of the *CreateDisplays* operation

5. Conclusions and Further work

In this paper we have described the implementation of an UML Profile in an UML CASE tool, using the *Presentation* profile as an example throughout the paper. During the process, we have sketched the limitations of the current UML CASE tools with respect to profile definition, specially when dealing with the constraints and derived elements of the profile, and explained some design strategies to avoid them. The rationale of our method is to extend the CASE tool with a set of profile operations that prevent an incorrect use of the profile and update the values of its derived elements. Finally, we have applied these strategies in the implementation of the profile within the Objecteering/UML tool.

We may conclude that, relating to profile definition, UML CASE tools still lack of maturity. In fact, the significant importance of profiles in conceptual modeling and other fields has not been accompanied by the necessary support of existing CASE tools. We think it is necessary a big effort from the UML CASE tool vendors in order to allow the software community easily develops and uses them.

As a further work, we would like to advance in the automation of our approach, developing a tool that given a profile definition as input generates, partially, the code of the operations needed to implement the profile in a specific CASE tool.

Acknowledgments

We would like to thank Dolors Costal, Antoni Olive, Maria Ribera Sancho and Ernest Teniente for their many useful comments to previous drafts of this paper. This work has been partially supported by the Ministerio de Ciencia y Tecnologia and FEDER under project TIC2002-00744.

References

[Argo03] ArgoUML, http://argouml.tigris.org/, visited March 2003.

[Cona99] Conallen, J; *"Modeling Web Application with UML"* http://www.conallen.com/whitepapers/webapps/ModelingWebApplications.htm

[Conc03] *Concept Base.* http://www-i5.informatik.rwth-aachen.de/CBdoc/, visited March 2003.

[HeKo01] Hennicker, R; Koch, N. *"Modeling the User Interface of Web Applications with UML",* UML 2001, LNI 7, pp.158-172.

[Jeck03] Jeckle, M; *"UML Tools (Case & Drawing)"*, http://www.jeckle.de/umltools.htm, visited March 2003.

[NaMa01] Naiburg, E.J; Maksimchuck, R.A. *"UML for Database Design"*, Ed. Addison-Wesley

[Obje03] Objecteering/UML. http://www.objecteering.com/us/produits.php, visited March 2003.

[ObyD03] Objects By Design; *"UML Modeling Tools"*,. http://www.objectsbydesign.com/tools/umltools_byProduct.html, visited March 2003

[OCLE03] Object Constraint Language Evaluator. http://lci.cs.ubbcluj.ro/ocle/, visited March 2003.

[OMG99] OMG.*"A UML Profile for CORBA"*, Version 1.0, August1999.

[OMG01a] OMG.*"Model Driven Architecture"*, http://www.omg.org/mda/presentations.htm. July 2001.

[OMG01b] OMG.*"Unified modeling Language Specification"*, Version 1.4, September 2001.

[OMG02a] OMG. "OMG XML Metadata Interchange Specification", Version 1.2, January 2002.

[OMG02b] OMG. "UML Profile for Enterprise Distributed Object Computing Specification", February 2002.

[OMG02c] OMG. "UML Profile for Schedulability, Performance and Time Specification", March 2002

[RuJB99] Rumbaugh, J.; Jacobson, I.; Booch, G. *"The Unified Modeling Language. Reference Manual"*, Addison-Wesley.

[Soft99] Softeam; *"UML Profiles and the J Language"*, White Paper, 1999, http://www.objecteering.com/pdf/us/uml_profiles.pdf, visited November 2002.

[Telo90] Mylopoulos, J.; Borgida, A.; Jarke, M.; Koubarakis, M. *"Telos: Representing Knowledge About Information Systems"*, ACM TOIS 8(4): 325-362 (1990).

[USE03] USE; http://www.db.informatik.uni-bremen.de/projects/USE/, visited March 2003.

Appendix A – *Presentation* profile constraints

R01: The height and width of a form must be positive.
context Form **inv**:
self.taggedValue->select(type.name='width').dataValue>0
and self.taggedValue->select(type.name='height').dataValue>0

R02: There cannot exists a cycle of associations with the stereotype *Displays*.
context Form **inv**:
not self.allDisplayedForms->includes(self)

R03: The coordinates x and y of the controls cannot be out of the range of the forms where they are placed inside and must be positive.
context Control **inv**:
self.taggedValue->select(type.name='coordinatex').dataValue>0
and self.taggedValue->select(type.name='coordinatey').dataValue>0
self.association.association->select(stereotype->one(name='Contains')). connection.participant->excludes(self)->
forAll (self.taggedValue->select(type.name='coordinatex').dataValue< taggedValue->select(type.name='width').dataValue
and self.taggedValue->select(type.name='coordinatey').dataValue< taggedValue->select(type.name='height').dataValue)

R04: The associations with the *Contains* stereotype must have two participants. One of them must be a form and the other must be a control.
context Contains **inv**:
self.connection->size()=2
and self.connection.participant->forAll(p | p.oclIsTypeOf(Class)) and self.connection.participant.stereotype->one(name='Form') and self.connection.participant.stereotype->one(name='EditBox' or name='Button')

R05: There cannot be two contains associations between the same form and control.
context Contains **inv**:
Association.allInstances->select(stereotype->one(name='Contains'))->forAll (a1,a2 | a1<>a2 implies
not a1.connection.participant->asSet()=a2.connection.participant->asSet())

R06: A *Displays* association must have two participants. Both participants must be forms. The navigability is single in the direction of the destination form.
context Displays **inv**:
self.connection->size()=2
and self.connection.participant->forAll(p | p.oclIsTypeOf(Class)) and self.connection->
select(isNavigable).participant.stereotype->one(name='Form') and self.connection->select(not isNavigable).participant.stereotype->one(name='Form')

R07: There cannot be two *Displays* associations between the same forms.
context Displays **inv**:
Association.allInstances->select(stereotype->one(name='Displays'))->
forAll (a1,a2 | a1<>a2 implies
not a1.connection.participant->asSet()=a2.connection.participant->asSet())

R08: The multiplicity of the associations with the *Contains* and *Displays* stereotype must be 1 in both association ends.

context Displays **inv**:
self.connection->select (multiplicity.range<>1 or multiplicity.range.lower<>1 or multiplicity.range.upper<>1)->isEmpty()

context Contains **inv**:
self.connection->select (multiplicity.range<>1 or multiplicity.range.lower<>1 or multiplicity.range.upper<>1)->isEmpty()

R09: The derived tag *ndisplayedforms* calculates the number of forms displayed by a form.

context Form **inv**:
self.taggedValue->select(type.name='ndisplayedforms').dataValue =
self.displayedForms->size()

The following additional operations are necessaries to define the above constraints:
The operation *displayedForms* returns a set containing all direct displayed forms.

context Form::displayedForms: Set(Class)
displayedForms=self.association->select(isNavigable).association->
select (stereotype->one(name='Displays')).connection.participant->
excludes(self)->asSet()

The operation *allDisplayedForms* returns a set containing all direct and indirect displayed forms.

context Form::allDisplayedForms:Set (Class)
allDisplayedForms= self.displayedForms->
union(self.displayedForms.allDisplayedForms)

Appendix B – Design and implementation of the profile operations

1. CreateButton

1.1 CreateButton operation design

context System:: CreateButton(coordx:Integer, coordy:Integer, type: ButtonType)
pre R03:
coordx>0 and coordy>0
post:
b.oclIsNew()
and b.oclIsTypeOf(Class)
and b.stereotype=Stereotype.allInstances->any(name='Button')
and taggedvaluecx.oclIsNew()
and taggedvaluecx.oclIsTypeOf(TaggedValue)
and taggedvaluecx.dataValue=coordx
and taggedvaluecx.type=TagDefinition.allInstances ->any(name='coordinatex')
and taggedvaluecx.modelelement =b
and taggedvaluecy.oclIsNew()
and taggedvaluecy.oclIsTypeOf(TaggedValue)
and taggedvaluecy.dataValue=coordy
and taggedvaluecy.type=TagDefinition.allInstances ->any(name='coordinatey')
and taggedvaluecy.modelelement =b
and taggedvaluetype.oclIsNew()
and taggedvaluetype.oclIsTypeOf(TaggedValue)
and taggedvaluetype.dataValue=type
and taggedvaluetype.type=TagDefinition.allInstances ->any(name='type')
and taggedvaluetype.modelelement =b

1.2 CreateButton operation implementation

```
void Class::CreateButton ()
{
JBox Form;
Stereotype S;
String scx;String scy; String btype;
int coordx;int coordy;
Object [] pCoordx; Object [] pCoordy; Object [] pBType;
//Creation of the dialog box
Form=createJBox("id1","Creation of a button",JLayoutVertical,false);
Form.addField("coordx","x position","",JControlTextField,150);
Form.addField("coordy","y position","",JControlTextField,150);
Form.addField("btype","button type","",JControlTextField,150);

if (Form.show())
{
  Form.getValue("coordx",scx);
  Form.getValue("coordy",scy);
  Form.getValue("btype",btype);
  coordx=scx.toInt();coordy=scy.toInt();
  if (coordx>0 and coordy>0 and (btype.strnequal("Accept",5) or btype.strnequal("Close",5)
```

```
        or btype.strnequal("Cancel",6)))   //pre R03
  {
    sessionBegin("createButton",true);
    S=this.findStereotypeInProject("Button","Class");
    //Adding the stereotype and the tags
    this.appendExtension(S);
    pCoordx.addElement(scx);
    this.createAndAddTaggedValueWithParams("","bcoordinatex",pCoordx,"");
    pCoordy.addElement(scy);
    this.createAndAddTaggedValueWithParams("","bcoordinatey",pCoordy,"");
    pBType.addElement(btype);
    this.createAndAddTaggedValueWithParams("","type",pBType,"");
    sessionEnd();
  }
  else infoUser("Warning","Coordinates must be higher than 0 and the type must be Accept, Cancel or Close");
}
Form.delete();
}   // method CreateButton
```

2. CreateContains operation

2.1 CreateContains operation design

context System:: CreateContains(f:Class, c:Class)
pre R03:
c.taggedValue->select(type.name='coordinatex').dataValue < f.taggedValue->select(type.name='width').dataValue
and c.taggedValue->select(type.name='coordinatey').dataValue < f.taggedValue->select(type.name='height').dataValue
pre R04:
f.stereotype->exists(name='Form')
and c.stereotype->exists(name='EditBox' or name='Button')
pre R05:
Association.allInstances->select(stereotype->exists(name='Contains') and connection.participant->asSet()=Set{f,c})->isEmpty()
post:
con.oclIsNew()
and con.oclIsTypeOf(Association)
and con.stereotype=Stereotype.allInstances->any(name='Contains')
and aef.oclIsNew()
and aef.oclIsTypeOf(AssociationEnd)
and aef.multiplicity.range->size()=1
and aef.multiplicity.range.lower=1
and aef.multiplicity.range.upper=1
and aec.oclIsNew()
and aec.oclIsTypeOf(AssociationEnd)
and aef.multiplicity.range->size()=1
and aec.multiplicity. range.lower=1
and aec.multiplicity. range.upper=1
and con.connection->first()=aef
and con.connection->last()=aec
and aef.participant=f

and aec.participant=c

2.2 CreateContains operation implementation

```
void Class::CreateContains ()
{

String NControl;
Class DestControl;
JBox Form;
NameSpace N;
AssociationEnd AE;
Stereotype S;
TagParameter tp;
int aux; int aux2;
TagParameter tp2;
boolean error;
boolean error2;
Association A;
Class[] CVec;

//Create the form
Form=createJBox("id1","Add a contains association",JLayoutVertical,false);
Form.addField("f1","Control name","",JControlTextField,200);

if (Form.show())
{
  Form.getValue("f1",NControl);
  //Obtain a class reference for the destination form
  N=this.OwnerNameSpace;
  DestControl=N.getClassByName(NControl);
  if(notVoid (DestControl))
  {
    if (this.ExtensionStereotype.Name=="Form" and (DestControl.ExtensionStereotype.Name=="Button"
or
        DestControl.ExtensionStereotype.Name=="EditBox"))  //pre R04
    {
      TagTaggedValue.<select(DefinitionTagType.Name=="width") //pre R03
      {
        getItemSet(ActualTagParameter,0,tp);
        DestControl.TagTaggedValue.<select(DefinitionTagType.Name=="ecoordinatex" or
DefinitionTagType.Name=="bcoordinatex")
        {
          getItemSet(ActualTagParameter,0,tp2);
           aux=tp.Value.toInt(); aux2=tp2.Value.toInt();
          if(aux<aux2) error=true;
        }
      }

      TagTaggedValue.<select(DefinitionTagType.Name=="height") //pre R03
      {
        getItemSet(ActualTagParameter,0,tp);
        DestControl.TagTaggedValue.<select(DefinitionTagType.Name=="ecoordinatey" or
DefinitionTagType.Name=="bcoordinatey")
        {
```

```
      getItemSet(ActualTagParameter,0,tp2);
      aux=tp.Value.toInt(); aux2=tp2.Value.toInt();
      if(aux<aux2)  error=true;
    }
  }

  //pre R05. No other contains association exists between the form and the control
  PartAssociationEnd.<select(RelatedAssociation.ExtensionStereotype.Name=="Contains")
  {
    RelatedAssociation.ConnectionAssociationEnd
    {
      if (OwnerClassifier==DestControl) error2=true;
    }
  }

  if (not error and not error2)
  {
    sessionBegin("contains",true);
AE=this.createAndAddMonoOrientedAssociation(DestControl,"","","1","1","1","1",Public,false,false);
    S=this.findStereotypeInProject("Contains","Association");
    A=AE.RelatedAssociation;
    A.appendExtension(S);
    sessionEnd();
  }else
    { if(error)  infoUser("Error","The control coordinates are higher than form dimensions");
      if(error2) infoUser("Error","There exists already a contains association");
    }
 }else infoUser("Error","The source must be a form and the destination a control");
}else infoUser("Error","The control does not exist");
}
Form.delete();
```

3. CreateDisplays operation

3.1 CreateDisplays operation design

context System:: CreateDisplays(sf:Class, df:Class)
pre R02:
not df.allDisplayedForms->includes(sf)
pre R06:
sf.stereotype->one(name='Form') and df.stereotype->one(name='Form')
pre R07:
Association.allInstances->select(stereotype->one(name='Displays') and
connection.participant->asSet()= Set{sf,df})->isEmpty()
post Create displays association:
dis.oclIsNew()
and dis.oclIsTypeOf(Association)
and dis.stereotype=Stereotype.allInstances->any(name='Displays')
and aesf.oclIsNew()
and aesf.oclIsTypeOf(AssociationEnd)
and aesf.isNavigable=true
and aesf.multiplicity.range->size()=1
and aesf.multiplicity.range.lower=1

```
and aesf.multiplicity.range.upper=1
and aedf.oclIsNew()
and aedf.oclIsTypeOf(AssociationEnd)
and aedf.isNavigable=false
and aedf.multiplicity. range.lower=1
and aedf.multiplicity.range->size()=1
and aedf.multiplicity. range.upper=1
and dis.connection->first()=aesf
and dis.connection->last()=aedf
and aesf.participant=sf
and aedf.participant=df
-- Induced effects ndisplayedforms:
and sf.taggedValue->select(type.name='ndisplayedforms').dataValue=
sf.displayedForms->size()
```

3.2 CreateDisplays operation implementation

```
void Class::CreateDisplays ()
{
String NForm;
Class DestForm;
JBox Form;
NameSpace N;
AssociationEnd AE;
Stereotype S;
TagParameter tp;
int aux;
boolean error;
Association A;
Class[] CVec;

//Create the form
Form=createJBox("id1","Add a displays association",JLayoutVertical,false);
Form.addField("f1","Form name","",JControlTextField,200);

if (Form.show())
{
  Form.getValue("f1",NForm);
  //Obtain a class reference for the destination form
  N=this.OwnerNameSpace;
  DestForm=N.getClassByName(NForm);
  if(notVoid (DestForm))
  {
    if (this.ExtensionStereotype.Name=="Form" and DestForm.ExtensionStereotype.Name=="Form")
//pre R06
    {
      //pre R07. No other displays association exists between both forms
      PartAssociationEnd.<select(RelatedAssociation.ExtensionStereotype.Name=="Displays")
      {
        RelatedAssociation.ConnectionAssociationEnd
        {
          if (OwnerClassifier==DestForm) error=true;
        }
      }
      if (not error)
```

```
      {
        sessionBegin("displays",true);
        AE=this.createAndAddMonoOrientedAssociation(DestForm,"","","1","1","1","1",Public,false,false);
        S=this.findStereotypeInProject("Displays","Association");
        A=AE.RelatedAssociation;
        A.appendExtension(S);

        //pre R02. Check that there is not a cycle in the displays associations
        ComputeDisplay(CVec);
       if(CVec.contains(this))
       {
         A.delete();infoUser("Error","There is a cycle between the forms");
       }
       else
       {  //Post Induced effects over ndisplayedforms
        TagTaggedValue.<select(DefinitionTagType.Name=="ndisplayedforms")
        {
          getItemSet(ActualTagParameter,0,tp);
          aux=tp.Value.toInt();
          aux=aux+1;
          tp.setValue(aux.toString());
        }
        sessionEnd();
       }
     }else infoUser("Error","There exists already a displays association between the forms");
   }else infoUser("Error","Both associations ends must be forms");
 }else infoUser("Error","The form does not exist");
}
Form.delete();

}   // method CreateDisplays


void Class::ComputeDisplay (inout Class[] _CVec)
{
AssociationEnd [] AEVec;
AssociationEnd [] AEVecAux;
Association Aux;
AssociationEnd AEAux;
//Direct associations
AEVec=this.PartAssociationEnd;
AEVec
{
  if (this.IsNavigable==true)
  {
    Aux=this.RelatedAssociation;
    if (Aux.ExtensionStereotype.Name=="Displays")
    {
      AEVecAux=Aux.ConnectionAssociationEnd.<select(IsNavigable==false);
      getItemSet(AEVecAux,0,AEAux);
      if(!_CVec.contains(AEAux.OwnerClass))
      {  _CVec.addElement(AEAux.OwnerClass);
         AEAux.OwnerClass.ComputeDisplay(_CVec);
      }
    }
```

```
  }
}
}   // method ComputeDisplay
```

4. CreateEditBox operation

4.1 CreateEditBox operation design

context System:: CreateEditBox(coordx:Integer, coordy:Integer)
pre R03:
coordx>0 and coordy>0
post:
e.oclIsNew()
and e.oclIsTypeOf(Class)
and e.stereotype=Stereotype.allInstances->any(name='EditBox')
and taggedvaluecx.oclIsNew()
and taggedvaluecx.oclIsTypeOf(TaggedValue)
and taggedvaluecx.dataValue=coordx
and taggedvaluecx.type=TagDefinition.allInstances ->any(name='coordinatex')
and taggedvaluecx.modelelement =e
and taggedvaluecy.oclIsNew()
and taggedvaluecy.oclIsTypeOf(TaggedValue)
and taggedvaluecy.dataValue=coordy
and taggedvaluecy.type=TagDefinition.allInstances ->any(name='coordinatey')
and taggedvaluecy.modelelement =e

4.2 CreateEditBox operation implementation

```
void Class::CreateEditBox ()
{

JBox Form;
Stereotype S;
String scx;String scy;
int coordx;int coordy;
Object [] pCoordx; Object [] pCoordy;
//Creation of the dialog box
Form=createJBox("id1","Creation of an editbox",JLayoutVertical,false);
Form.addField("coordx","x position","",JControlTextField,150);
Form.addField("coordy","y position","",JControlTextField,150);

if (Form.show())
{
  Form.getValue("coordx",scx);
  Form.getValue("coordy",scy);
  coordx=scx.toInt();coordy=scy.toInt();
  if (coordx>0 and coordy>0)   //pre R03
  {
    sessionBegin("createEditBox",true);
    S=this.findStereotypeInProject("EditBox","Class");
    //Adding the stereotype and the tags
    this.appendExtension(S);
    pCoordx.addElement(scx);
```

```
    this.createAndAddTaggedValueWithParams("","ecoordinatex",pCoordx,"");
    pCoordy.addElement(scy);
    this.createAndAddTaggedValueWithParams("","ecoordinatey",pCoordy,"");
    sessionEnd();
   }
  else infoUser("Warning","Coordinates must be higher than 0");
 }
Form.delete();
}   // method CreateEditBox
```

5. CreateForm operation

5.1 CreateForm operation design

context System:: CreateForm(wid:Integer, hei:Integer)
pre R01:
wid>0 and hei>0
post:
f.oclIsNew()
and f.oclIsTypeOf(Class)
and f.stereotype=Stereotype.allInstances->any(name='Form')
and taggedvaluewid.oclIsNew()
and taggedvaluewid.oclIsTypeOf(TaggedValue)
and taggedvaluewid.dataValue=wid
and taggedvaluewid.type=TagDefinition.allInstances ->any(name='width')
and taggedvaluewid.modelelement =f
and taggedvaluehei.oclIsNew()
and taggedvaluehei.oclIsTypeOf(TaggedValue)
and taggedvaluehei.dataValue=hei
and taggedvaluehei.type=TagDefinition.allInstances ->any(name='height')
and taggedvaluehei.modelelement =f
-- Induced effects ndisplayedforms:
and taggedvaluendf.oclIsNew()
and taggedvaluendf.oclIsTypeOf(TaggedValue)
and taggedvaluendf.dataValue=0
and taggedvaluendf.type=TagDefinition.allInstances ->
any(name='ndisplayedforms')
and taggedvaluendf.modelelement =f

5.2 CreateForm operation implementation

```
void Class::CreateForm ()
{
JBox Form;
Stereotype S;
String sh;String sw;
int height;int width;
Object [] pHeight; Object [] pWidth; Object [] pNDisp;
//Creation of the dialog box
Form=createJBox("id1","Creation of a new form",JLayoutVertical,false);
Form.addField("width","Width","",JControlTextField,150);
Form.addField("height","Height","",JControlTextField,150);
```

```
if (Form.show())
{
  Form.getValue("width",sw);
  Form.getValue("height",sh);
  width=sw.toInt();height=sh.toInt();
  if (height>0 and width>0)   //pre R01
  {
    sessionBegin("createForm",true);
    S=this.findStereotypeInProject("Form","Class");
    //Adding the stereotype and the tags
    this.appendExtension(S);
    pHeight.addElement(sh);
    this.createAndAddTaggedValueWithParams("","height",pHeight,"");
    pWidth.addElement(sw);
    this.createAndAddTaggedValueWithParams("","width",pWidth,"");
    pNDisp.addElement("0");
    this.createAndAddTaggedValueWithParams("","ndisplayedforms",pNDisp,"");
    sessionEnd();
  }
  else infoUser("Warning","Height and width must be higher than 0");
}
Form.delete();
}   // method CreateForm
```

6. ModifyCoordx operation

6.1 ModifyCoordx operation design

context System:: ModifyCoordx(c:Class, coordx:Integer)
pre cIsAControl**:**
c.stereotype->exists(name='Button' or name='EditBox')
pre R03:
coordx>0
and Association.allInstances->select(stereotype->exists(name='Contains') and connection.participant->includes(c)).connection.participant->excludes(c)->forAll(taggedValue->select(type.name='width').dataValue>coordx
post:
c.taggedValue->select(type.name='coordinatex').dataValue=coordx

6.2 ModifyCoordx operation implementation

```
void Class::ModifyCoordx ()
{
JBox Form;
String scx;
int coordx; int aux;
TagParameter tp;
boolean error;

//Creation of the dialog box
Form=createJBox("id1","Modify the x coordinate",JLayoutVertical,false);
```

```
Form.addField("coordx","Coordinate x","",JControlTextField,150);

if (Form.show())
{
  Form.getValue("coordx",scx);
  coordx=scx.toInt();
  if (coordx>0 and (ExtensionStereotype.Name=="Button" or ExtensionStereotype.Name=="EditBox"))
//pre R01
  {
    //pre R03. Check the new coordinate
    PartAssociationEnd.<select(RelatedAssociation.ExtensionStereotype.Name=="Contains")
    {
      RelatedAssociation.ConnectionAssociationEnd
      {
        if (OwnerClassifier.ExtensionStereotype.Name=="Form")
        { //pre R03
          OwnerClassifier.TagTaggedValue.<select(DefinitionTagType.Name=="width")
          {
            getItemSet(ActualTagParameter,0,tp);
            aux=tp.Value.toInt();
            if(coordx>aux) error=true;
          }
        }
      }
    }

    if(not error)
    {
      sessionBegin("modifycoordx",true);
      TagTaggedValue.<select(DefinitionTagType.Name=="ecoordinatex" or
DefinitionTagType.Name=="bcoordinatex") //pre R03
      {
        getItemSet(ActualTagParameter,0,tp);
        tp.setValue(scx); //Update the tag with the new value
      }
      sessionEnd();
    }else infoUser("Error","There is a conflict with the form dimensions");
  }else infoUser("Error","The class is not a control or the new coordinate is <0");
}
Form.delete();
}   // method ModifyCoordx
```

7. ModifyCoordy operation

7.1 ModifyCoordy operation design

context System:: ModifyCoordy(c:Class, coordy:Integer)
pre cIsAControl**:**
c.stereotype->exists(name='Button' or name='EditBox')
pre R03:
coordx>0
and Association.allInstances->select(stereotype->exists(name='Contains') and
connection.participant->includes(c)).connection.participant->excludes(c)->

forAll(taggedValue->select(type.name='height').dataValue>coordy
post:
c.taggedValue->select(type.name='coordinatey').dataValue=coordy

7.2 ModifyCoordy operation implementation

```
void Class::ModifyCoordy ()
{
JBox Form;
String scy;
int coordy; int aux;
TagParameter tp;
boolean error;

//Creation of the dialog box
Form=createJBox("id1","Modify the y coordinate",JLayoutVertical,false);
Form.addField("coordy","Coordinate y","",JControlTextField,150);

if (Form.show())
{
  Form.getValue("coordy",scy);
  coordy=scy.toInt();
  if (coordy>0 and (ExtensionStereotype.Name=="Button" or ExtensionStereotype.Name=="EditBox"))
//pre R01
  {
    //pre R03. Check the new coordinate
    PartAssociationEnd.<select(RelatedAssociation.ExtensionStereotype.Name=="Contains")
    {
      RelatedAssociation.ConnectionAssociationEnd
      {
        if (OwnerClassifier.ExtensionStereotype.Name=="Form")
        { //pre R03
          OwnerClassifier.TagTaggedValue.<select(DefinitionTagType.Name=="height")
          {
            getItemSet(ActualTagParameter,0,tp);
            aux=tp.Value.toInt();
            if(coordy>aux) error=true;
          }
        }
      }
    }

    if(not error)
    {
      sessionBegin("modifycoordy",true);
      TagTaggedValue.<select(DefinitionTagType.Name=="ecoordinatey" or
DefinitionTagType.Name=="bcoordinatey") //pre R03
      {
        getItemSet(ActualTagParameter,0,tp);
        tp.setValue(scy); //Update the tag with the new value
      }
      sessionEnd();
    }else infoUser("Error","There is a conflict with the form dimensions");
  }else infoUser("Error","The class is not a control or the new coordinate is <0");
}
```

```
Form.delete();
}   // method ModifyCoordy
```

8. ModifyHeight operation

8.1 ModifyHeight operation design

context System:: ModifyHeight(f:Class, hei:Integer)
pre fIsAForm**:**
f.stereotype->exists(name='Form')
pre R01:
hei>0
pre R03:
Association.allInstances->select(stereotype->exists(name='Contains') and connection.participant->includes(f)).connection.participant->excludes(f)->forAll(taggedValue->select(type.name='coordinatey').dataValue<hei
post:
f.taggedValue->select(type.name='height').dataValue=height

8.2 ModifyHeight operation implementation

```
void Class::ModifyHeight ()
{
JBox Form;
String sh;
int height; int aux;
TagParameter tp;
boolean error;

//Creation of the dialog box
Form=createJBox("id1","Modify the height tag",JLayoutVertical,false);
Form.addField("height","Height","",JControlTextField,150);

if (Form.show())
{
  Form.getValue("height",sh);
  height=sh.toInt();
  if (height>0 and this.ExtensionStereotype.Name=="Form")   //pre R01
  {
    //pre R03. Check the new height with each control contained inside the form
    PartAssociationEnd.<select(RelatedAssociation.ExtensionStereotype.Name=="Contains")
    {
      RelatedAssociation.ConnectionAssociationEnd
      {
        if (OwnerClassifier.ExtensionStereotype.Name=="Button" or
OwnerClassifier.ExtensionStereotype.Name=="EditBox")
        { //pre R03
          OwnerClassifier.TagTaggedValue.<select(DefinitionTagType.Name=="ecoordinatey" or
DefinitionTagType.Name=="bcoordinatey")
          {
            getItemSet(ActualTagParameter,0,tp);
            aux=tp.Value.toInt();
            if(height<aux) error=true;
```

```
        }
      }
    }
  }

  if(not error)
  {
    sessionBegin("modifyheight",true);
    TagTaggedValue.<select(DefinitionTagType.Name=="height") //pre R03
    {
      getItemSet(ActualTagParameter,0,tp);
      tp.setValue(sh); //Update the tag with the new value
    }
    sessionEnd();
  }else infoUser("Error","There is a conflict with a control coordinates");
 }else infoUser("Error","The class is not a form or the height is <0");
}
Form.delete();
}   // method ModifyHeight
```

9. ModifyWidth operation

9.1 ModifyWidth operation design

context System:: ModifyWidth(f:Class, wid:Integer)
pre fIsAForm**:**
f.stereotype->exists(name='Form')
pre R01:
wid>0
pre R03:
Association.allInstances->select(stereotype->exists(name='Contains') and connection.participant->includes(f)).connection.participant->excludes(f)->forAll(taggedValue->select(type.name='coordinatex').dataValue<wid
post:
f.taggedValue->select(type.name='width').dataValue=wid

9.2 ModifyWidth operation implementation

```
void Class::ModifyWidth ()
{
JBox Form;
String sw;
int width; int aux;
TagParameter tp;
boolean error;

//Creation of the dialog box
Form=createJBox("id1","Modify the width tag",JLayoutVertical,false);
Form.addField("width","Width","",JControlTextField,150);

if (Form.show())
{
```

```
 Form.getValue("width",sw);
 width=sw.toInt();
 if (width>0 and this.ExtensionStereotype.Name=="Form")   //pre R01
 {
   //pre R03. Check the new width with each control contained inside the form
   PartAssociationEnd.<select(RelatedAssociation.ExtensionStereotype.Name=="Contains")
   {
     RelatedAssociation.ConnectionAssociationEnd
     {
       if (OwnerClassifier.ExtensionStereotype.Name=="Button" or
OwnerClassifier.ExtensionStereotype.Name=="EditBox")
       { //pre R03
         OwnerClassifier.TagTaggedValue.<select(DefinitionTagType.Name=="ecoordinatex" or
DefinitionTagType.Name=="bcoordinatex")
         {
           getItemSet(ActualTagParameter,0,tp);
           aux=tp.Value.toInt();
           if(width<aux) error=true;
         }
       }
     }
   }

   if(not error)
   {
     sessionBegin("modifywidth",true);
     TagTaggedValue.<select(DefinitionTagType.Name=="width") //pre R03
     {
       getItemSet(ActualTagParameter,0,tp);
       tp.setValue(sw); //Update the tag with the new value
     }
     sessionEnd();
   }else infoUser("Error","There is a conflict with a control coordinates");
 }else infoUser("Error","The class is not a form or the width is <0");
}
Form.delete();
}   // method ModifyWidth
```

10. RemoveDisplays operation

10.1 RemoveDisplays operation design

context System:: RemoveDisplays(d:Association)
pre dIsADisplay:
d.stereotype->exists(name='Displays')
post:
let
sf:Class=d.connection->select(isNavigable).participant
df:Class=d.connection->select(not isNavigable).participant
in
Association.allInstances->select(stereotype->exists(name='Displays')) =
Association@pre.allInstances->select(stereotype->exists(name='Displays'))->
excludes(d)

```
-- Induced effects ndisplayedforms
and sf.taggedValue->select(type.name='ndisplayedforms').dataValue=
sf.taggedValue->select(type.name='ndisplayedforms').dataValue@pre-1
```

10.2 RemoveDisplays operation implementation

```
void Association::RemoveDisplays ()
{
TagParameter tp;
int ndisp;

if (ExtensionStereotype.Name=="Displays")
{
  sessionBegin("removedisplays",true);
  ConnectionAssociationEnd.<select(IsNavigable==true)
  {
   OwnerClassifier.TagTaggedValue.<select(DefinitionTagType.Name=="ndisplayedforms")
   {
    getItemSet(ActualTagParameter,0,tp);
    ndisp=tp.Value.toInt();
    ndisp=ndisp-1;
    tp.setValue(ndisp.toString());
   }
  }
  this.delete();
  sessionEnd();
}
else infoUser("Error","It must be a displays association");
}   // method RemoveDisplays
```