ADVANCED PROBLEM TYPE DEVELOPMENT: DATA DEPENDENCIES AND GUI CUSTOMIZATION (TKWIDGET)

Jorge Suit Pérez Ronda, Enrique Escolano Tercero, Miguel Pasenau de Riera

26th January 2004

International Center for Numerical Methods in Engineering (CIMNE)

gid@cimne.upc.es

Abstract

This paper covers two advanced features in the problem type development using **GiD**: `DEPENDENCIES` and `TKWIDGET`. Using those features the developer gains more control on the look and feel of the data windows and build a more customized and professional GUI of the problem type interface.

1 INTRODUCTION

When defining a problem type the developer specifies the data interface through the definitions files `.mat`, `.cnd` and `.prb`. Regarding the complexity of this definition we had classified the problem type as "level I" (simple) or "level II" (advanced).

At the "level I" every data (condition, material, interval or general data) is defined as a collection of properties or questions (labeled as `QUESTION`), a value (labeled as `VALUE`) for every `QUESTION` and, possibly, some attributes modifying the `QUESTION` or the visual structure of the associated window. At this level a standard interface is obtained and no extra programming is needed, but this result is generally useful in an academic environment or as a prototype. For an extended description of the **GiD**'s problem types and its definition the reader is referred to [2, 3].

Thus, if we want to develop a more user specific and complex interface we must resort to other technique and deal with Tcl/Tk programming language using **GiD** Tcl/Tk extension. [1]. This is what we call "level II" problem type development.

Here we are concerned with two features used when developing a "level II" problem type namely `DEPENDENCIES` and `TKWIDGET`. Although `DEPENDENCIES` does not require programming in Tcl/Tk it is considered as "level II" because it modifies the default data window behavior.

Of course these are not the only features used at this level, there are others such as: menu & toolbar customization, HTML support, **GiD**'s control function. All this can be seen at [2, 3].

From now on we assume the reader is familiar with "level I" and Tcl/Tk programming. The rest of the paper is structured as follow: first, in section 2, DEPENDENCIES are explained, next TKWIDGET is detailed in section 3. Concluding remarks are given in the last section.

2 DATA DEPENDENCIES

Consider you want to develop a problem type where both 2D and 3D properties can be handled by the user depending on the type of domain selected. In such a case it would be worthy to have only visible those properties corresponding to the selected domain. The previous requirement can be implemented with what we called DEPENDENCIES. With DEPENDENCIES we can stablish relations among questions and change the STATE (NORMAL, HIDDEN, DISABLED) and VALUE of other QUESTION depending on the value of the QUESTION having the DEPENDENCIES defined.

DEPENDENCIES are defined as question's attributes and they enable us to stablish relations, through actions, among questions not only in the same data set (condition, material, interval data or general data) but also among questions of different data sets.

One or more actions can be defined for every value of the QUESTION. An action takes effect on another QUESTION or TITLE of the window where the dependence is defined. The QUESTION for which the DEPENDENCIES are defined is knows as *source* QUESTION while the one receiving the action is known as *target* QUESTION.

The syntax is the following:

```
QUESTION: Your_Label
VALUE: The_Value
DEPENDENCIES: (v0, ACTION00, REF00, v00,...,ACTION0n, REF0n, v0n)
DEPENDENCIES: (v1,...)
...
DEPENDENCIES: (vk,...)
```

where:

- vi is the value that trigger the actions ACTIONij
- REFij is the target that could be a TITLE reference or a QUESTION reference, and will be referred as *lvalue*.
- vij is the new value for the reference REFij, and will be referred as *rvalue*.

The previous defines a set of actions to be triggered whenever the QUESTION takes the given value, for instance the set of actions ACTION00, ..., ACTION0n are executed when the QUESTION Your_Label takes the value v0. A special value #DEFAULT# can be specified in the last dependence meaning that the set of actions associated will be triggered whenever the QUESTION takes a value not listed in the previous dependencies.

Actions The actions operate over *lvalues*. Following are the actions' names:

- `TITLESTATE` : change the state of the `TITLE` given as *lvalue*.
- `SET` : change the value and set the state of the `QUESTION` to disabled.
- `HIDE` : this action hide the `QUESTION` and change the value to the new value.
- `RESTORE` : this enable or restore the state of the `QUESTION` to normal and change the value to the new value.

References *(lvalues)* An *lvalue* is a `TITLE` or `QUESTION` reference . A `TITLE` *lvalue* should be the name of a `TITLE` attribute while a `QUESTION` *lvalue* could be:

- a local reference: a `QUESTION` name in the same data set
- a global reference: [`conditions`.]condname.question, [`materials`.]matname.question, `gendata`.question[1]. Where condname is the name of a condition, matname is the name of material and question is the name of a question within the condition, material or problem data.

Using a global reference we can change the value of a `QUESTION` in other data set[2]. Examples of global references are:

```
conditions.Line_Constraints.X_constraint
Line_Constrains.X_constraint
```

New values (*rvalue*) When the action is `TITLESTATE` the possible values are `NORMAL`, `DISABLED`, `HIDDEN`. In other case *rvalue* is the value to be assigned to the *lvalue* and could be a constant value or the value of other `QUESTION`. A constant value is a string without spaces and any separator used in `DEPENDENCIES` such as: , ( ). *rvalue* could also be the special value `#CURRENT#` meaning not to change the value, and in that case only the state of the *rvalue* is changed depending on the action applied.

For *lvalues* of types `#CB#` and `#MAT#` the *rvalue* must be one of the possible selections or a new definition of the list of values, for instance the following will change the list's contents for the combo box `Target` to `(one,two,three)` if the value of `Source` is `letters` but if the value is `digits` the list's contents is changed to `(1,2,3)`.

```
QUESTION: Source#CB#(letters,digits)
VALUE: letters
DEPENDENCIES: (letters,RESTORE,Target,"#CB#(one,two,three)") \
              (digits,RESTORE,Target,"#CB#(1,2,3)")
QUESTION: Target#CB#(1,2,3)
VALUE: 1
```

[1] Interval Data references are not implemented yet

[2] this data set does not need to be visible

```
CONDITION: Steel_section
CONDTYPE: over lines
CONDMESHTYPE: over body elements
QUESTION: Local_Axes#LA#(-Default-,-A
VALUE: -Default-
QUESTION: SteelName
VALUE: IPN-80
QUESTION: SteelType
VALUE: A37
END CONDITION
```

Figure 1: default layout for data windows

A value of another `QUESTION` can also be used as *rvalue*, in that case the syntax is `->qreference` where `qreference` is a local or global reference to another `QUESTION`, for instance:

```
QUESTION: Source#CB#(1,2)
VALUE: 1
DEPENDENCIES: (1,SET,Target,->Another_Q1) \
              (#DEFAULT#,RESTORE,Target,#CURRENT#)
QUESTION: Target
VALUE: 1
...
QUESTION: Another_Q1
VALUE: Any_Value
```

3 GUI CUSTOMIZATION: TKWIDGET

As seen previously, using `DEPENDENCIES`, the developer can implement some constraints among questions. But the look and layout of the window associated to the data definition is the default provided for the "level I" problem type, and it can not be the most suitable for every application.

The developer can change the way a `QUESTION` is displayed and if he wishes he can also change the whole contents of the window, maintaining the basic behavior of the data set, i.e. in condition window: assign, unassign, draw; in material window: create material, delete material and so on.

With the default layout for the data windows the questions are placed one after another in one column inside a container frame, the question's label in column zero and the `VALUE` in column one, see an example in figure 1

The developer can override this behavior using `TKWIDGET`. `TKWIDGET` is defined as an attribute of a `QUESTION` and the value associated to it must be the name of Tcl procedure, normally implemented in a Tcl file of the problem type. This procedure will take care of drawing the `QUESTION`[3] and also attending some events related to the window and its data.

The prototype of a `TKWIDGET` procedure is as follow:

```
proc TKWidgetProc {event args} {
```

[3] A `TKWIDGET` may also draw the entire contents of the window

```
  switch $event {
    INIT {
       ...
    }
    SYNC {
       ...
    }
    DEPEND {
       ...
    }
    CLOSE {
       ...
    }
  }
}
```

The procedure should return:

- empty string "" meaning that every thing was OK.
- a two list element `{ERROR-TYPE Description}` where `ERROR-TYPE` could be `ERROR` or `WARNING`. `ERROR` means that something is wrong and the action should be aborted. If `ERROR-TYPE` is `WARNING` then the action is not aborted but `Description` is showed as a message. In any case if `Description` is not empty a message is showed.

The argument `event` is the type of event and `args` is the list of arguments depending on the event type. The possible events are: `INIT`, `SYNC`, `CLOSE` and `DEPEND`.

INIT this event is triggered when **GiD** needs to display the corresponding `QUESTION` and the list of arguments is `{frame row-var GDN STRUCT QUESTION}`: `frame` is the container frame where the widget should be placed, `row-var` is the name of the variable, used by **GiD**, with the current row in the frame, `GDN` and `STRUCT` are the names of internal variables needed to access the values of the data, `QUESTION` is the `QUESTION's` name for which the `TKWIDGET` procedure was invoked. Normally the code for this event should initialize some variables and draw the widget.

SYNC triggered when **GiD** requires a synchronization of the data. Normally it involves updating some of the QUESTIONs of the data set. The argument list is `{GDN STRUCT QUESTION}`.

CLOSE triggered before closing the window, as mentioned this can be canceled if an `ERROR` is returned from the procedure.

DEPEND this event is triggered when a dependence is executed over the `QUESTION` for which the `TKWIDGET` is defined, ie., that `QUESTION` is an *lvalue* of the dependence. The list of arguments is `{GDN STRUCT QUESTION ACTION value}` where `GDN`, `STRUCT` and `QUESTION` are as before, `ACTION` could be `SET`, `HIDE` or `RESTORE` and `value` is the value assigned in the dependence.

```
CONDITION: Steel_section
CONDTYPE: over lines
CONDMESHTYPE: over body elements
QUESTION: Local_Axes#LA#(-Default-,-A
VALUE: -Default-
QUESTION: SteelName
VALUE: -
QUESTION: SteelType
VALUE: -
TKWIDGET: SteelSections
END CONDITION
```

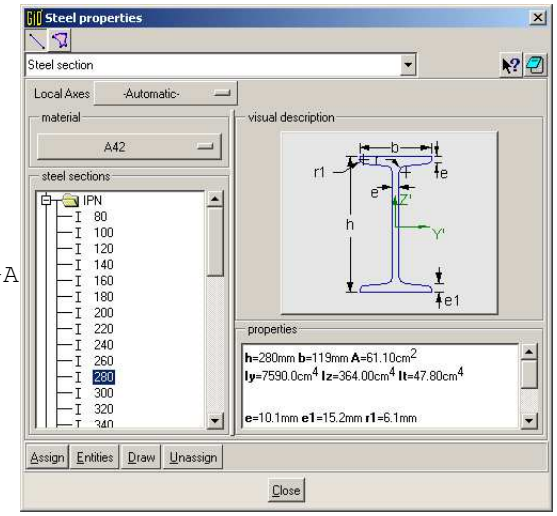

Figure 2: `TKWIDGET` example

Figure 2 shows a fragment of the data definition file and the `GUI` as result. This sample is taken from RamSeries [5] and in this case the `TKWIDGET` is used to create the whole contents of the condition windows (note de difference with figure 1)

4 CONCLUSIONS

"Level II" problem types interface is being developed in order to provide a professional look like any other commercial simulation software. They has been used successfully in software products developed as **GiD**'s problem types such as: **RamSeries**[5] (Structural Analysis), **TDyn**[6] (Fluid Dynamic) , **GiD-NASTRAN** interface[4] and **Stampack**[7] (sheet stamping and forming).

Both `DEPENDENCIES` and `TKWIDGET` (among other **GiD** features) has enabled building **GiD**'s problem types with a sophisticated and customized GUI in accordance to the final user needs. Future work includes to provide more control on the behavior of the data window in such a way that the problem type developer can intercept events such as condition assign/unassign, material assign/unassign with the corresponding list of entities associated to the event. We think that this will complement the set of features found in "level II".

References

[1] BRENT WELCH, KEN JONES, J. H. *Practical Programming in Tcl and Tk*, 4th ed. Prentice Hall PTR, june 2003.

[2] CIMNE. Gid online support. http://www.gidhome.com/support.

[3] CIMNE. *GiD Reference Manual Version 7*, 2003.

[4] COMPASS S.A. GiD-NASTRAN interface. http://www.compassis.com/en/productos/nastran-interface/index.html.

[5] COMPASS S.A. Ram Series. http://www.compassis.com/en/productos/ram-series/index.html.

[6] COMPASS S.A. Tdyn. http://www.compassis.com/en/productos/tdyn/index.html.

[7] QUANTECH ATZ. Stampack. http://www.quantech.es/product-stampack.htm.