

[0029] The step of obtaining the value of the attribute of the second class may further comprise interpreting the position information to extract the relative location of the second class; and retrieving the reference to the second class by recursively navigating the hierarchical arrangement of classes in accordance with the interpreted position information.

[0030] The computer program may be written in the Perl language, in which case the class-making module may be the Class::MethodMaker module.

[0031] Additionally, each class declaration may additionally include a privilege level identifier associated with each attribute thereof. In this case the method may further comprise comparing the privilege level identifier with a predetermined user associated privilege level and only allowing access to that attribute if the user privilege level is at least as high.

[0032] In a further embodiment, there is provided a method, within an object-oriented computer program, of creating a dependency between a first class and an element of a second class in a hierarchical arrangement of classes created by a class-making module based on declarative definitions of each class. The method may comprise defining, within the first class definition, position information defining the relative position within the hierarchy of the element of the second class, and rule information defining the nature of the dependency; such that an accessor method created for accessing the attribute of the first class is arranged for: obtaining the value of the attribute of the second class; and modifying the attribute of the first class in accordance with the value of the attribute of the second class and the information contained in the second information item.

[0033] The invention will now be described, by way of non-limiting example, with reference to the accompanying diagrams, in which:

[0034] FIG. 1 is a diagram showing a configuration tree representation of a simple computer system;

[0035] FIG. 2 is a flow diagram outlining the main functional steps of a tree navigation function according to an embodiment;

[0036] FIG. 3 is a diagram showing an extract of the configuration tree of FIG. 1;

[0037] FIG. 4 is a diagram showing an expanded extract of the configuration tree of FIG. 1;

[0038] FIG. 5a is a flow diagram outlining the main functional steps for implementing a value dependency according to an embodiment;

[0039] FIG. 5b is a flow diagram outlining the main functional steps for accessing a value which has dependencies according to an embodiment;

[0040] FIG. 6 is a flow diagram outlining the main functional steps for implementing a value dependency using a Perl-type implementation; and FIG. 7 is a flow diagram outlining the main functional steps for implementing an object dependency using a Perl-type implementation.

[0041] Referring again to FIG. 1, there is shown an example computer system made up of the following nodes:

a computer 102, a video card 104, a monitor 108, and a processor 114. The computer node 102 has an attribute of computer type, 103, video card has attributes of video memory, 106, and pixel clock, 107, the monitor has attributes of screen resolution, 110, and refresh rate, 112, and the processor has attributes of clock frequency, 118, and cache size, 120. Each of the attributes and nodes may have an associated behavior. For example, the computer type attribute 103 may be limited to taking a value of 'laptop' or 'desktop', the attribute clock frequency 116 may have maximum and minimum values and so on.

[0042] As previously described, one way to create a computer program to model the configuration behavior of a system is to use an object oriented approach and to define classes which model the behavior of each of the system elements.

[0043] Using a generic class-making module, for example such as the Perl Class::MethodMaker module, the various classes required for modeling the configuration behavior of the different system elements may be declared as shown below. For ease of explanation the tabulated views, below, outline the nature of each class, and the pseudo-code representations outline an example high-level implementation of the class which may be implemented in a number of different programming languages.

COMPUTER CLASS - TABULATED VIEW		
CLASS NAME: Computer		
ATTRIBUTES	Type	Comments
computer_type	enum	Choice: 'Laptop', 'Desktop'
video_card	VideoCard	
Processor	Processor	

[0044]

```

COMPUTER CLASS - PSUEDO-CODE
class Computer ; # declaration of the main computer class
{
  attributes: # define the attributes of the class
  {
    computer__type :
    {
      type: enum;
      choice: ('desktop', 'laptop');
    }
    video__card: # define the other nodes which depend from the
    Computer
    # node
    {
      type VideoCard ; # video__card is of type VideoCard class
    }
    processor: #
    {
      type Processor ; # processor is of type Processor class
    }
  }
}

```