dependent on one or more attributes of one or more other classes. For example, referring back to the example shown in **FIG. 1**, it may be that the refresh rate attribute **112** is computed from the screen resolution attribute **110** and the pixel clock attribute **107**. In such a case, it may not be necessary that the screen resolution attribute be automatically updated each time one of the other attributes is changed. Rather, it may be preferable that the value of the refresh rate attribute is computed each time its value is retrieved. In this case, known hereinafter as a compute value, the screen resolution attribute within the monitor class class declaration may be expressed as:

```
screen_resolution: # define the characteristics of the
                                variable screen_resolution
{
        type : compute; # is of type compute object
                                # variables:
        {
          name: 'rate', location: ..\refresh_rate ;
          name: 'clock', location: ..\..\pixel_clock;
        };
        compute_formula: 'rate / clock'
}
```

[0091] Thus, each time the accessor method is called to retrieve the value of the attribute screen_resolution, the compute object (which is the attribute type of screen_resolution) will retrieve, for example using the aforementioned tree navigation functionality, the values of the variables specified, and will apply the compute formula specified to determine the value of the attribute screen_resolution.

[0092] In a further embodiment of the present invention, each attribute in the class declaration may be assigned an identifier indicating a privilege level required in order to be able to access the attribute. This may be useful, for example, for preventing a casual user from making modifications to system critical configuration parameters by effectively 'hiding' any such attributes. For example, a number of different levels of privilege such as a 'user', 'support', 'expert', etc. may be defined. If a user is identified as a 'support' user, only attributes having a privilege level of 'user' or 'support' will be accessible. Any attempt to access attributes having a privilege level above 'support' will result in an error and will be refused.

[0093] Using the techniques described herein, a configuration model for a system can be quickly and easily implemented using, for example, declarative, rather than hard-coded, class implementations. Such a system may be used for populating the attributes of the different system elements, and ensuring that account is taken of any attributes or objects which are dependent on any other attribute or object. This may be useful, for example, by allowing a default attribute value to be specified based on the value of another attribute value.

[0094] Such a configuration tree model may be used, with an appropriate interface, to enable configuration parameters for a configuration tree to be created and verified. Such an interface may include, for example, a command line type interface in which a user enters configuration details, for example by specifying an object and an attribute value. Alternatively, an interface may be provided which reads an existing configuration file and which populates the configuration tree using the values read from the configuration file. This permits the integrity of the configuration file to be verified helping ensure the validity of the configuration parameters contained therein. Preferably, such a system enables a configuration file to be created by writing to a configuration file the attribute values associated with each object in the configuration tree.

[0095] In Perl, for example, use may be made of the built in tie mechanisms which can be used to 'hide' much of the complex functionality from the user, thereby enabling the class declarations to remain relatively simple. For example, the Perl tied mechanisms provide a convenient way of allowing the implementation and behavior of built in data types to be altered, which can be used to implement the above-described functionality without introducing unnecessary complication into the computer program.

[0096] Those skilled in the art will appreciate that the techniques described with reference to the various embodiments described above may be implemented in various different manners and in a variety of different programming languages. It will also be appreciated that one way in which the above describes techniques can be provided is in the form of an article of manufacture comprising a program storage medium having computer readable program code, for example, for use on a general purpose computer system.

1. 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 using declarative definitions of each class,

the method comprising:

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; and incorporating functionality within the first class to interpret the rule and position information to create the dependency.

2. A method according to claim 1, wherein the element of the second class is an attribute, and wherein the rule and position information is associated with an attribute of the first class, the incorporated functionality being arranged for performing the steps of:

obtaining the value of the attribute of the second class using the position information; and

determining a value of the attribute of the first class by using the obtained value and the rule information.

3. A method according to claim 2, wherein the step of obtaining the value of the attribute of the second class further comprises:

interpreting the position information to extract the relative location of the second class;

retrieving the reference to the second class by recursively navigating the hierarchical arrangement of classes in accordance with the interpreted position information.

4. A method according to claim 2 or **3**, wherein the step of obtaining the value further comprises: