

program. For example, the parameters of a modem may have to change depending on the geographical location of the computer. Thus, if the geographical location is represented by an attribute of a class, the attributes of a modem class may be need to change accordingly is the geographical location attribute changes.

[0013] Such behavior dependencies can be provided by hard-coding the dependency information in each of the appropriate classes and by linking attributes of one class with attributes of another. However, this can lead to program code which is cumbersome and hence lead to increased maintenance difficulties. Additionally, should any of the dependencies change, this will generally require modification of each of the classes concerned. Furthermore, a configuration system with a large number of system elements will generally require the computer program to have a large number of classes. In many cases, however, many of the classes will be similar in nature which can lead to the class definition section of the computer program being long and repetitive in nature.

[0014] One way of reducing the coding effort required is to define the properties of each class using a declarative description, and to use a generic class-making module for creating class implementations using the class declaration. This is particularly useful where many similar but different classes are required, as is often the case when modeling configuration trees.

[0015] In the Perl programming language, for example, there exist a number of generic class-making routines such as the `Class::Struct` module, which forms part of the standard Perl distribution, and the well-known `Class::MethodMaker` module which is available through the comprehensive Perl archive network (CPAN) at <http://www.cpan.org>. `Class::MethodMaker` may be thought of as a function which takes input parameters which define the nature of the data containers (or attributes) and the class accessor methods for each class, and which creates a class implementation based on those input parameters.

[0016] A computer program written using such techniques therefore comprises two principle sections: a first section containing the class declarations that define the class implementations which will be created at run-time; and a second section containing program code which will create the configuration model in, for example, the memory of a computer, by referencing the class implementations created at run-time by the class-making module or, alternatively, by calling the classes through the methods created at run-time.

[0017] In this way, the classes may be defined in a simple and concise declarative manner which often helps in facilitating the initial writing of the code as well as improving the maintainability of the code.

[0018] However, such a declarative approach does not enable dependencies between classes or attributes (other than those inherent in the tree structure due to the hierarchical structure) to be defined.

[0019] Accordingly, one aim of the present invention is to overcome at least some of the above-mentioned problems by providing a framework which enables dependency information to be easily defined using a declarative approach to class definition.

[0020] According to a first embodiment of the present invention, 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 using declarative definitions of each class. The method comprises 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.

[0021] In one embodiment, where the element of the second class is an attribute, and the rule and position information is associated with an attribute of the first class, the incorporated functionality is 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.

[0022] Suitably, 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.

[0023] The step of obtaining the value may further comprise registering, with the second class, the rule information and a reference to the attribute of the first class such that an accessor method used to interrogate the attribute of the second class modifies, by way of the registered reference, the attribute of the first class by applying the registered rule information in association with the attribute of the second class.

[0024] Preferably the incorporated functionality within the first class is provided through one or more external classes having functionality for interpreting the position and rule information.

[0025] In a second embodiment, where the element of the second class is an attribute, and where the rule information defines the creation of an object from a choice of classes based on the value of the attribute, the method may further comprise obtaining the value of the attribute of the second class using the position information; and creating a new object in accordance with the obtained value and the rule information.

[0026] The method may further comprise creating a hidden object intermediate a first object created from the first class and the new object for regulating access to the first object.

[0027] The step of obtaining the value may further comprise registering, with the second class, a reference to the hidden object and the rule information such that an accessor method used to interrogate the attribute of the second class may cause the hidden object to create a further new object, and hence to prevent access to the previous created object, in accordance with the rule information.

[0028] Where a further new object is created, the method may further comprise copying attribute values from the previous created object to the further new object on a best-effort basis.