

## BRIEF DESCRIPTION OF THE DRAWINGS

[0013] FIG. 1 illustrates the validation of a computer application having class and interface definitions.

[0014] FIG. 2 is a conceptual block diagram illustrating the validation of a program at the language-independent level and the language-dependent level.

[0015] FIG. 3A is a flow diagram illustrating the validation of a computer program implemented using a compiler-language section.

[0016] FIG. 3B is a flow diagram illustrating the validation of a computer program implemented using a compiler-language section and a script code section.

[0017] FIG. 4 illustrates the validation of a computer program having a compiler-language section and a script code section.

[0018] FIG. 5 illustrates the use of definition and implementation modules to validate a program and generate Hypertext Markup Language (HTML) code.

Like reference numbers and designations in the various drawings indicate like elements.

## DETAILED DESCRIPTION

[0019] As shown in FIG. 1, in one implementation of the invention, tree structures are used to represent a computer program. These representations are used to generate and validate one or more executable programs, as will be described. The representations can include a tree structure defining the implementation modules, in this particular case, classes to be implemented 100, and a tree structure defining the definition modules, in this particular case, the associated interfaces 105. It is advantageous to define the implementation modules and the definition modules using a language-independent representation. A language-independent representation is not restricted to any particular programming language used to implement an executable program. In one exemplary implementation, the implementation modules and the definition modules can be specified in a language that can be used as a meta-language, e.g., Extensible Markup Language (XML). Extended Backus-Naur Form (EBNF), Abstract Syntax Notation One (ASN.1), or LISP.

[0020] The XML descriptions of the interfaces and classes can be used to generate executable programs, for example, by using an Extensible Stylesheet Language (XSL) style sheet 115 in conjunction with an XSL processor 110 to generate program implementation code 120. The executable programs are language-dependent representations specific to a particular programming language used to implement the programs. XSL includes two parts: a language for transforming XML documents, and an XML vocabulary for specifying formatting semantics. An XSL style sheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary. Other technologies, e.g., the Apache Velocity template engine, can be used in place of XSL.

[0021] The program described by the interfaces and classes can be validated at two levels. On the language-independent or XML level, a syntax check is performed for the interface description and the implementation class

description. On the language-dependent level, to the extent that the generated implementation code can be compiled, the interface and class definitions can be validated by compiling the code. In this case, the compiler 125 verifies the class and interface definitions by performing usage and implementation checks. Usage checks verify that a customer class only uses interfaces that have been promised by a provider class. Implementation checks verify that the implementation of a provider class provides all the promised interfaces.

[0022] FIG. 2 is a conceptual block diagram illustrating the validation of a program at the language-independent level and the language-dependent level. As described above, a definition module 200 and an implementation module 205 representing the program are received. In one implementation, the computer program can be represented using more than one definition module 200 and/or more than one implementation module 205. The implementation module 205 defines the classes to be implemented by the program and the definition module 200 defines the associated interfaces. The validation at the language-independent level is performed as a syntax check for the definition module 200 and the implementation module 205. The implementation module 205 is used to generate the classes 215 and the definition module 200 is used to generate the associated interfaces 210. Language-dependent validation of the classes 215 and interfaces 210 generated from the language-independent descriptions can be performed by compiling the classes 215 and the interfaces 210 using a compiler 125. The compiler 125 verifies the class and interface definition by performing usage and implementation checks on the language-dependent representation of the classes 215 and the interfaces 210.

[0023] FIG. 3A is a flow diagram illustrating the validation of a computer program implemented using a compiler-language section. A language-independent description of a computer program is received (step 300). As described above, the language-independent description can include implementation modules 205 and definition modules 200 describing the computer program. The language-independent description is validated (step 310) and a syntax check is performed for the implementation modules 205 and the definition modules 200 (step 312). The language-independent description is used to generate a language-dependent program (step 320), e.g., by generating a program implementation using a particular programming language. The language-dependent program is validated (step 330) and usage and semantics checks are performed by compiling the generated interfaces and classes (step 332).

[0024] FIG. 3B is a flow diagram illustrating the validation of a computer program implemented using a compiler-language section and a script language section. As discussed above, a language-independent representation of a computer program is received (step 300), and the language-independent representation is validated (step 310) by performing a syntax check for the implementation modules 205 and definition modules 200 (step 312). A language-dependent representation of the program is generated (step 320), and the generated language-dependent representation is validated (step 330). The generated computer program includes a compiler-language section and a script code section. As described above, the validation of the generated compiler-language section includes usage and semantics checks performed by compiling the generated interfaces 210 and